DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING, SECOND CYCLE, 30 CREDITS

STOCKHOLM, SWEDEN 2019

Network Drone Control using Deep Reinforcement Learning

ALEX HERMANSSON GROBGELD

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Network Drone Control using Deep Reinforcement Learning

ALEX HERMANSSON GROBGELD

Master in Machine Learning
Date: October 1, 2019
Supervisor: Hamid Faragardi
Examiner: Elena Troubitsyna
School of Electrical Engineering and Computer Science
Host company: Ericsson
Swedish title: Kontroll av Drönare i Nätverk med Deep Reinforcement Learning

Abstract

In this work, a reinforcement learning approach is adopted to control a drone in a cellular network. The goal is to find paths between arbitrary locations such that low radio quality areas, defined with respect to signal-to-interference-plus-noise-ratio, are avoided with the cost of longer flight paths. The controlling agent learns to take decisions without having access to any propagation model, learning only through feedback in the form of a reward signal evaluating its behavior. In the reward function, designed for this particular problem, there are simple parameters for modifying the focus of the agent to put more emphasis on short paths, or paths with high radio quality.

The proposed agent uses a learning algorithm that combines Double Deep Q-Networks with Hindsight Experience Replay to handle the stochastic environment with multiple goals. A neural network is used to approximate the optimal Q-values and the experiences are collected using a Boltzmann exploration policy.

Three different scenarios are studied: flight trajectories on constant altitude, with and without measurement noise in the radio quality, and flight trajectories on varying altitudes with measurement noise. In all scenarios, simulation results show that the agent successfully avoids low radio quality areas by taking longer flight paths, as desired. The probability of flying through areas with low radio quality is reduced by between 62 and 75 percent compared to the baseline agent, that flies greedily toward the goal. In 90 percent of the evaluation instances, for all three scenarios, the flight paths are shorter than two times the shortest possible path and the median length is around 1.3 times the shortest path. Thus, the reinforcement learning agent holds a clear advantage over the baseline for applications were the radio quality is of high importance.

In the case of constant flight altitude, it is also possible to visualize and gain insight into the decision making process through the learned value function. It is evident that this function reflects the radio quality as one can see patterns resembling those of the underlying signal distribution.

Sammanfattning

I detta arbete används reinforcement learning for att styra en drönare i ett cellulärt nätverk. Målet är att hitta rutter mellan godtyckliga positioner, samtidigt som områden med låg radiokvalitet, med avseende på signal-to-interference-plus-noise-ratio, undviks till kostnad av längre flygsträckor. Den styrande agenten lär sig att ta beslut utan att ha tillgång till någon propageringsmodell, enbart genom återkoppling i form av en belöningssignal som utvärderar dess agerande. I belöningsfunktionen som är konstruerad för detta specifika problem, finns enkla parametrar som kan användas för att modifiera ett önskat fokus på korta flygsträckor eller rutter med hög radiokvalitet.

Den föreslagna agenten använder en inlärningsalgoritm som kombinerar Double Deep Q-Networks med Hindsight Experience Replay för att hantera den stokastiska miljön med varierande målpositioner. Ett neuralt nätverk används för att approximera de optimala Q-värdena och erfarenheterna samlas in med en Boltzmann utforskningspolicy.

Tre olika scenarier studeras: flygrutter på konstant altitud, med och utan mätningsbrus i radiokvaliteten samt flygrutter på varierande altitud med mätningsbrus. I alla scenarier visar simuleringsresultaten att agenten med framgång lär sig att undvika områden med låg radiokvalitet genom att flyga längre rutter, vilket är det önskade beteendet. Sannolikheten att flyga genom områden med låg radiokvalitet reduceras med mellan 62 och 75 procent jämfört med en basnivåagent som flyger raka vägen till målpositionen. I 90 procent av evalueringsfallen, för alla tre scenarier, är flygrutterna kortare än två gånger den kortast möjliga rutten och medianen ligger kring 1.3 gånger den kortast möjliga rutten. Alltså är agenten klart fördelaktig i jämförelse med basnivån i applikationer där radiokvaliten är högst betydande.

I fallet med konstant altitud är det möjligt att visualisera och få insikt i beslutsprocessen genom den inlärda värdefunktionen. Det är tydligt att denna funktion reflekterar radiokvaliteten eftersom man kan se mönster som liknar de som finns i den underliggande signaldistributionen.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Machine Learning . . . 2
- 1.3 This Study . . . 2
- 1.4 Research Questions . . . 3
- 1.5 Research Methodology . . . 3
- 1.6 Outline . . . 3

2 Related Work **5**
- 2.1 Why Reinforcement Learning? . . . 5
- 2.2 Value Based Methods . . . 5
- 2.3 Policy Gradient Methods . . . 8

3 Background **10**
- 3.1 Reinforcement Learning . . . 10
  - 3.1.1 Markov Decision Processes . . . 11
  - 3.1.2 Return, Policy, and Objective . . . 12
  - 3.1.3 Value Functions . . . 13
  - 3.1.4 Monte Carlo and Temporal-Difference methods . . . 14
  - 3.1.5 Q-Learning . . . 15
  - 3.1.6 Exploration . . . 16
- 3.2 Neural Networks . . . 17
  - 3.2.1 Multilayer Perceptron . . . 17
  - 3.2.2 Activation functions . . . 17
  - 3.2.3 Loss Functions and Optimizers . . . 18
- 3.3 Deep Reinforcement Learning . . . 19
  - 3.3.1 Deep Q-Networks . . . 19
  - 3.3.2 Double Q-Learning and Double DQN . . . 20

3.3.3 Multiple Goals & Universal Value Function Approximators . . . 21
3.3.4 Hindsight Experience Replay . . . 22
3.4 A Measure of Radio Quality . . . 24

4 Method **25**
4.1 Problem Definition . . . 25
4.2 Experimental setup . . . 26
4.2.1 Environment . . . 26
4.2.2 Markov Decision Process . . . 28
4.3 Learning Algorithm . . . 31
4.4 Evaluation . . . 34
4.4.1 Baseline Agent . . . 34
4.4.2 Metrics . . . 34

5 Results & Analysis **39**
5.1 Constant Altitude Without Noise . . . 39
5.2 Constant Altitude With Noise . . . 42
5.3 Varying Altitude With Noise . . . 46
5.4 Ablation Study . . . 48

6 Discussion **51**
6.1 Results Discussion . . . 51
6.2 Validity . . . 52
6.2.1 Construct Validity . . . 52
6.2.2 Internal Validity . . . 52
6.2.3 Conclusion Validity . . . 53
6.3 Sustainability and Ethical Aspects . . . 53

7 Conclusion **54**
7.1 Summary . . . 54
7.2 Future Work . . . 55

Bibliography **56**

Chapter 1

Introduction

1.1 Motivation

The fifth generation cellular network, 5G, is expected to be deployed in the near future and with it comes a wide range of potential applications. The 5G networks will offer high data rates, reduced latency, and support for massive device connectivity [1]. One of many possible applications is long-range autonomous drones that would benefit greatly from the high connectivity and better data rates to provide functionality such as industrial inspection, delivery, and surveillance services among others. The forecasts for commercial drone usage predict that it will constitute a $100 billion market over the coming years [2].

One idea is to use the existing terrestrial mobile network for drone applications. Some use cases might require drone to drone communication and others might need communication between the drones and some station on the ground, commonly with a need for high throughput. A potential issue is that the existing network is designed for usage on a low altitude, meaning that the radio quality can be quite poor in some areas. If the applications allow, the drone itself can optimize the radio quality by avoiding such areas autonomously.

The interest in the drone market is especially high owing to these kinds of autonomous system applications, which naturally motivates research within artificial intelligence and machine learning, the foundations of autonomy.

1.2 Machine Learning

Machine learning is the study of algorithms that learn to perform certain tasks from data instead of being explicitly instructed as in traditional algorithms. Often, it boils down to learning an approximate function from the data to perform the task [3]. There are three main branches of machine learning, where the most common is *Supervised Learning*. In this setting, a model learns to map input to output from a labeled data set with examples of input-output pairs. This can be to predict a set of real valued numbers, as in regression, or a category, as in classification, from a set of features of the input. *Unsupervised Learning* tries to find latent structure in a data set, for example by clustering similar data points together or by modeling the generative distribution. In this setting there are no labels, which is why it is referred to as unsupervised. The third setting, that is the focus in this thesis, is *Reinforcement Learning*. This is the branch that deals with sequential decision making under uncertainty. The goal is to learn a policy for how to act optimally, given some information of the surroundings, referred to as the state of the environment.

Together with other machine learning tools such as deep learning, reinforcement learning has become a method for attacking many different kinds of problems. Recent success stories include robotic control [4], mastering the game of Go [5], and reducing energy consumption in large data centers [6]. In this setting where reinforcement learning is combined with deep learning, the field is generally called *Deep Reinforcement Learning*.

1.3 This Study

Maintaining high radio quality will be crucial for certain industrial and commercial applications of drones that are connected through a cellular network. Another important factor for an increased efficiency of such applications is autonomy. In this study, these two issues are tackled through experiments with data coming from a radio network simulator provided by Ericsson. The objective is to construct a method that can control a drone between any two locations while avoiding areas of low radio quality. It is also of interest to easily be able to extend the method if further complexity is introduced such as having multi-

ple users and therefore having different signal dynamics. A method based on reinforcement learning is proposed and tested in several different cases that include flight trajectories in two and three dimensions along with static and dynamic radio quality distributions. The learned control policy is compared with a baseline that takes the shortest path toward the goal location. From the results, it is clear that the learned control policy is far superior to the baseline policy when radio quality is to be kept above a certain threshold.

1.4 Research Questions

Can deep reinforcement learning be used for learning to control a drone in a cellular network so that areas of low radio quality are avoided. Can such a method be kept general so that it is easy to extend when adding more complexity to the problem.

1.5 Research Methodology

The general methodology of this work is outlined in Figure 1.1. First the research questions were formulated, a pre-study was made, and a baseline was defined. After that, a deeper study of related work was conducted in order to propose potential solutions. These solutions were thereafter implemented and evaluated against the baseline, then this cycle was iterated.

1.6 Outline

The thesis is divided into seven chapters. This chapter gave a motivation and introduction to the topic of network drone control, a brief overview of machine learning, together with the research questions and methodology. Chapter 2 summarizes related work that is connected to the reinforcement learning techniques used in this work and answers the question of why it is proposed as a solution to the task. After that, in chapter 3, a more in depth review of the theory of reinforcement learning, neural networks, and the combination of the former two are presented along with a description of a radio quality metric called signal-to-interference-plus-noise-ratio. In chapter 4, the experimental

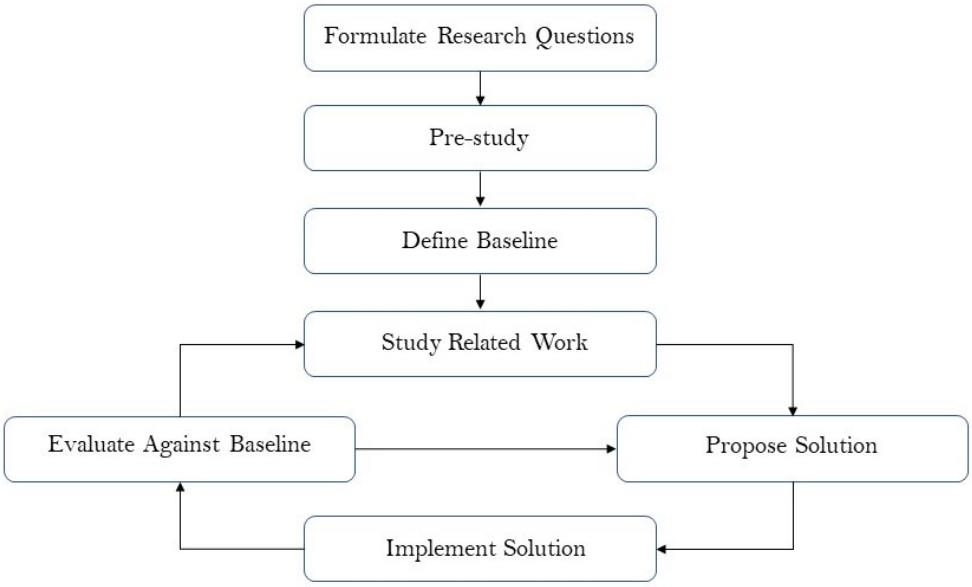

Figure 1.1: Research Methodology

setup, proposed solution, and the evaluation approach are explained. Chapter 5 displays the results of three experiments and analysis associated with each of those. There is also an assessment of the different components of the learning algorithm - the proposed solution. In chapter 6, there is a more general analysis of the results, discussions about the validity of the results, and sustainability and ethical aspects. The last chapter, chapter 7 concludes the thesis and proposes some future work that might be worth investigating.

Chapter 2

Related Work

2.1 Why Reinforcement Learning?

Simple path planning tasks in a grid with obstacles have traditionally been solved using search methods such as *Dijkstra* or *A** [7]. When further constraints are added, the problem is instead often posed as an optimization problem where approximation schemes have to be developed. It can be due to some factor that is inherently stochastic in the environment or because of a large state space. In fact, when the position is continuous the state space cannot even be enumerated, thus making search methods impossible to use. The issue with both search and optimization methods is that they rely on accurate models of the environment, which are not always available. Problems that are hard to model explicitly, but where data can be attained without too much effort, are well suited for reinforcement learning. In this setting, a learning agent is trying to accomplish some objective by interacting with the environment, without having access to the environment dynamics [8]. A scalar reward signal can be explicitly provided by the environment, like in games, or implicitly so that the reward function has to be designed for the objective. This is the most likely case in a real world application.

2.2 Value Based Methods

Many methods in reinforcement learning are based on values of states and actions. When these values are directly used in order to derive a

policy, the method is referred to as a *value based* method. Two classical examples are the *Sarsa* [9] and *Q-learning* [10, 11] algorithms. In these methods, estimates of the values of state-action pairs are stored in a table and updated during the learning process. There is no need for a model of the environment dynamics in comparison to most optimization techniques. However, the fact that the values are stored in tables is quite limiting since there has to be a finite set of states (also they cannot be too many because of memory requirements), and also all states have to be visited a large amount of times for meaningful learning to occur.

To overcome this limitation, one idea is to use function approximation to learn the values instead. In this way, the agent should be able to generalize and take decisions in unseen scenarios. A lot of recent success has come from using neural networks as function approximations. This forms the basis of the *Deep Q-Network* (DQN) which combines the learning process of Q-learning with a deep neural network as a function approximator to compute action values [12]. To stabilize learning, the authors also introduces two additional ideas. The first, called experience replay, is to use a memory that is filled with experiences that can be reused for learning. When sampling from the memory it breaks the correlation between transitions. It is important since inputs to a neural network are expected to be independent and identically distributed. The second idea is to use an additional neural network, called a target network, to compute bootstrapped targets. The parameters of the network are kept fixed for a number of iterations, and then set equal to the weights of the network that is continuously updated. This is so that the target distribution is only changing slowly, which is required for stable training of neural networks.

One improvement of DQN is *Prioritized Experience Replay* where the replay memory is modified to prioritize samples where the expected learning outcome is as high as possible [13]. This provides a faster, more sample efficient learning process.

Another improvement is the *Double DQN* that improves the performance of DQN by reducing bias in the action-value estimator [14]. The reduction of bias in the estimator was shown earlier by the same author for Q-learning, and was extended to hold for DQN. When Double DQN was introduced, it improved the state of the art on many benchmarks and it also has the benefit of being simple to implement.

While the improvements above are made on the learning algorithm, the *Dueling DQN* further increases the performance by introducing a

neural network architecture better suited for reinforcement learning [15]. Instead of having a network with one output stream corresponding to the action value, the dueling architecture has two output streams that correspond to the advantages and the value of a state. These two streams are then combined to form the action values. This is shown to generalize the learning, and can easily be incorporated into several learning algorithms.

Additionally, there are more improvements that, combined with the above mentioned ones, are referred to as *Rainbow* [16]. This combination of techniques provide current state-of-the-art results on many benchmarks in the discrete action setting, for example on the Atari 2600 testbed.

Some attempts have been made to explicitly incorporate planning in the policies through the use of model-based reinforcement learning. In the previously discussed model-free methods, no model of the environment is learned, rather the agent learns to optimize the cumulative reward directly. On the other hand, in model-based methods, the agent also learns about the transition dynamics and reward function of the environment through interaction. Thus, planning can be executed using this approximation of the world providing simulated experiences that result in increased sample efficiency. One such planning method is called *Dyna-Q*, which is an integrated architecture for learning, planning, and reacting [17]. The method was introduced as a tabular method, meaning it can only handle small countable state spaces, but has later been extended for use with function approximation similar to the extension of Q-learning to DQN. In [18], this method is referred to as *Deep Dyna-Q* (DDQ) and is used to train a dialog agent that learns to answer simple questions after only experiencing 100 user interactions.

In some environments, there are multiple goals that the policy of the agent can be conditioned on. Instead of shaping a reward function to make the agent get closer to the desired goal in a particular episode, *Hindsight Experience Replay* (HER) is one successful way to deal with this [19]. With this approach, a replay buffer has to be used (as in DQN) and the reward function has to be available to the agent. The episodes are, in hindsight, replayed with the goal set to one of the states that were actually achieved during interaction in order to more often provide a reinforcement signal to learn from. The technique has been used to provide very fast learning of difficult robotics tasks such as pushing,

sliding and pick-and-place of objects both in simulation and deployed on a physical robot. Not only was the learning fast, but these tasks have never been solved using other reinforcement learning methods.

2.3 Policy Gradient Methods

Reinforcement learning algorithms that learn a policy directly without the need of learning action values are called *Policy Gradient* methods. Here the policy is parameterized directly by a neural network instead of inferring the policy from the action values. One advantage of these algorithms is the possibility of learning to act when the action space is continuous, where one example is robotic manipulation in which torques are given to the joints [4].

This whole class of methods is based on the *Policy Gradient Theorem* that provides an elegant way to compute the gradient of the expected return with respect to the parameters of the policy [8]. This is quite a powerful result, since one would expect the gradients to be dependent on the dynamics of the environment.

One of the first policy gradient methods was the REINFORCE algorithm presented in [20]. Most recent actor-critic methods build upon this algorithm, and provide ways to stabilize learning such as with *Trust Region Policy Optimization* (TRPO) [21], and *Proximal Policy Optimization* [22] (PPO). In these two papers the objective is modified from the expected return to also include a penalty for deviation from the current policy, thus motivating the updated policy to be kept within a proximity, or trust region, of the previous one. This leads to less unstable behaviour, that is otherwise often prevalent in policy gradient methods.

The *Value Iteration Network* is an architecture for policy gradient methods, similarly to how the Dueling DQN introduced a specific reinforcement learning network architecture for value based methods. It approximates the dynamic programming algorithm *Value Iteration* using a convolutional neural network module inside the network [23]. This is shown to provide implicit planning capabilities so that the agent can generalize to very different scenarios than it has experienced during the learning phase.

There have also been successful attempts to parallelize the learning through asynchronous updates of the policy network by provid-

ing gradients from agents acting in different instances of the learning environment. One such parallellized algorithm is called *Asynchronous Advantage Actor Critic* (A3C) and can lead to significant learning speed-ups on simulated domains such as the Atari 2600 testbed [24].

A comparison of some of the algorithms discussed above are given in table 2.1.

Algorithm	**Value/Policy**	**-Policy**	**Model-**	**Empirical Results**
Rainbow [16]	Value	Off	Free	State-of-the-art result on the Atari 2600 testbed.
DDQ [18]	Value	Off	Based	Dialog agent that learns a model of user behaviour.
HER [19]	Value	Off	Free	Learning of difficult robotic tasks.
PPO [22]	Policy gradient	On	Free	Teaching complex simulated robots to walk and run.
A3C [24]	Policy gradient	On	Free	Substantial learning speed-up on the Atari 2600 testbed.

Table 2.1: A comparison of different reinforcement learning algorithms

Chapter 3

Background

3.1 Reinforcement Learning

This section describes some of the main concepts in reinforcement learning important for this thesis. For the most part, statements are taken from the book *Reinforcement Learning, An Introduction* [8] if not referenced otherwise.

The reinforcement learning problem is essentially a sequential decision making problem. An agent interacts with an environment by choosing actions based on the environment's current state (or an observation of the state), then it receives feedback in terms of a scalar reward and a new observation of the environment state. The reward and new state are stochastically determined by the dynamics of the environment, which in general are not known to the agent. The goal for the agent is to try and maximize the sum of all rewards, the return, by finding a good way to act, a policy. This process of interaction is repeated over time and is visualized in Figure 3.1.

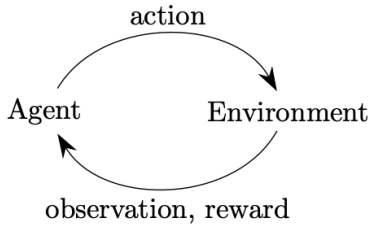

Figure 3.1: Agent-environment interaction [25]

3.1.1 Markov Decision Processes

Mathematically, one can describe the interaction between the agent and the environment as a *Markov Decision Process* (MDP) where the interaction happens along a sequence of discrete time steps, $t = 0, 1, 2, 3, \ldots$. An MDP is defined by the following components:

1. A state space $\mathcal{S}$, which is the set of possible states of the environment.

2. An action space $\mathcal{A}$, which is the set of actions that can be chosen by the agent at each time step.

3. The dynamics of the environment,

$$p(s', r|s, a) = \Pr\{S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a\}, \tag{3.1}$$

   which is a probability distribution over next state s' and reward r given current state s and action a.

If both state space and action space are finite spaces, with a finite number of elements, the process is called a *finite MDP*. The fact that the dynamics are conditioned only on the current state and action, not previous ones, means that the Markov property holds, which is where the name comes from. In the full reinforcement learning problem, the dynamics are not available to the agent. If they were, the optimal behaviour could be derived using dynamic programming [26].

There are two classes of reinforcement learning algorithms, those that explicitly try to learn the environment dynamics; *model-based* methods and those that learn to act without a model of the dynamics; *model-free* methods.

During interaction, at each time step t, the agent receives a representation of the environment's state $S_t = s$ and on that basis selects an action $A_t = a$. The next time step, the agent receives a scalar reward $R_t = r$ and the state of the environment is now changed to $S_{t+1} = s'$, which are jointly determined by the dynamics p. The outcome of these random variables, (s, a, r, s'), is referred to as an *experience tuple*. The interaction between the agent and the environment can also be summarized with a *trajectory*:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots \tag{3.2}$$

In this thesis, only *episodic tasks* are considered, meaning that the tasks have a clear end in some finite time $t = T > 0$, which is not necessarily the same in each episode.

3.1.2 Return, Policy, and Objective

Informally, the objective of the agent is to maximize cumulative reward in the long run. Formally, one often defines the *return*, denoted G_t, as a function of the reward sequence from time step t, and then seek to maximize this in expectation. The most common case is the *discounted return*, which is defined by

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{K} \gamma^k R_{t+k+1} \tag{3.3}$$

where $\gamma \in [0, 1]$ is a parameter called the *discount rate* serving two purposes. First of all, it is mathematically appealing since it causes the return, equation (3.3), to always take finite values, even in the case of continuing tasks where $K \to \infty$. Secondly, it can be used to change the behaviour of the agent to either be more farsighted, by using larger values of γ, or myopic, by using smaller values of γ. In the extreme case when $\gamma = 0$, the agent only cares about the next reward. In the other extreme case when $\gamma = 1$, the agent cares equally about all rewards until the end of its lifetime.

The return, G_t, can be written in terms of the return of a later time step, for example as

$$\begin{aligned} G_t &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} \cdots) \\ &= R_{t+1} + \gamma G_{t+1}. \end{aligned} \tag{3.4}$$

This is an important relationship that is exploited in many reinforcement learning algorithms that uses bootstrapping (for example Q-learning, section 3.1.5).

The objective of the agent is to find a policy, $\pi : \mathcal{S} \mapsto \mathcal{A}$ or often written $\pi(a|s)$, that maximizes the expectation of G_t for any t. The policy is a function that maps from state (or observation) to action at each time step. It can either be stochastic, such that the policy provides a probability distribution over possible actions for every state, or it can be deterministic so that the policy directly maps to an action of choice.

The actual learning in the algorithms refers to how the agent's policy is changed with experience gained from interacting with the environment.

3.1.3 Value Functions

Value functions, of different sorts, are used in many algorithms to estimate how good it is to be in a certain state, or how good it is to take a particular action from a certain state. It is defined in terms of expected return, and with respect to a particular policy.

First, the *state-value function* of a policy π, denoted $v_\pi(s)$, is defined as the expected return when starting in s and then acting according to π thereafter,

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s \right], \quad \forall s \in \mathcal{S}. \tag{3.5}$$

In a similar vein, the *action-value function* of a policy π, denoted $q_\pi(s, a)$ is defined as the expected return when starting in s, taking action a, then acting according to π thereafter,

$$q_\pi(s, a) = \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right], \quad \forall s \in \mathcal{S},\ \forall a \in \mathcal{A}. \tag{3.6}$$

Note that the action a taken at time t does not have to come from the policy, but all actions after do.

Another related value function that is sometimes used is the *advantage function*. It is defined as the difference of the action-value function and the state-value function,

$$a_\pi(s, a) = q_\pi(s, a) - v_\pi(s), \quad \forall s \in \mathcal{S},\ \forall a \in \mathcal{A}. \tag{3.7}$$

Intuitively, it describes how good an action is, compared to the average or one can think of it as measuring the relative advantage of the actions.

For either of these value functions, we refer to the function corresponding to the optimal policy as the optimal value function. For example, the optimal action-value function is given by

$$q_*(s, a) = \max_\pi q_\pi(s, a). \tag{3.8}$$

3.1.4 Monte Carlo and Temporal-Difference methods

Value functions can be estimated in several ways, here we describe two such methods and briefly discuss what lies between them. For the state-value function, one simple approach is to keep track of the return that comes after visiting a state, when following a certain policy. This type of method is called a *Monte Carlo* method. If many samples like this are collected and averaged, by the law of large number, this estimate will converge to the true value function since it is an expectation. Another example of a Monte Carlo method is the following. Let the estimate of v_π be denoted with V, then for a state S_t, we can move the estimate towards G_t with a step-size of α by doing updates

$$V(S_t) \leftarrow V(S_t) + \alpha \left[G_t - V(S_t) \right]. \tag{3.9}$$

In any kind of Monte Carlo method, we need to have full access to the return, G_t, and therefore we must wait until the episode terminates. Since, we have to wait until termination for only one update, it can take quite a long time for a Monte Carlo method to converge.

Another approach, for which the estimate also converges to the true value, can often be much more sample efficient (needing less interaction until convergence). These types of algorithms are called *Temporal-Difference* (TD) methods, and instead of waiting until the final outcome to do updates of the value-function, they bootstrap from current estimates. The simplest TD method uses the relation in equation (3.4) and makes the update

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right] \tag{3.10}$$

right after transitioning to S_{t+1} and receiving R_{t+1}. This method is called *one-step* TD since it only takes one step before updating the value of a state. There are also slightly more complicated versions where n steps are taken, called *n-step* TD. When looking at n-step methods, we see that Monte Carlo and one-step TD are closely related. In the Monte Carlo method, the estimate of v_π is updated towards the discounted return:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T. \tag{3.11}$$

In *one-step* TD, the estimate of v_π is updated towards the immediate reward plus the discounted estimated value of the next state (which in itself is an estimate of G_t):

$$G_{t:t+1} = R_{t+1} + \gamma V_t(S_{t+1}). \tag{3.12}$$

In between those two special cases, we have the n-step TD method, which after n-steps estimates the rest of the discounted return using the value function:

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n}). \tag{3.13}$$

3.1.5 Q-Learning

The previous section describes methods for estimating the state-value function, which is usually referred to as the *prediction problem*. However, it does not tell us how to act optimally, or how to derive the optimal policy - the *control problem*. Most methods use some variation of *generalized policy iteration* in which the value function is estimated for an initial policy, then the policy is improved such that it is strictly better than the previous policy with respect to expected return. This process is repeated until the policy stabilizes, which happens when it is optimal in the case of a finite MDP [8]. In order to solve the control problem, we generally estimate the action-value function instead of the state-value function. One very popular control algorithm is *Q-learning*, a method to directly approximate the optimal action-value function. When the estimate is identical to this function, by taking actions with the highest q-values, we act optimal. The estimate function, Q, is updated according to

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]. \tag{3.14}$$

The update moves the estimate $Q(S_t, A_t)$ closer toward $R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$ which is a bootstrapped version of the discounted return. This is similar to one-step TD described in section 3.1.4, but now with the action-value function instead of the state-value function.

The fact that the return is estimated using the *greedy* action, note the max operator in equation (3.14), and not the action taken by the *behaviour* policy makes the algorithm an *off-policy* algorithm. In off-policy learning, there is a behaviour policy for collecting experience and there is a *target* policy which is evaluated and improved. In the

case of Q-learning, the target policy is a greedy policy since it selects actions deterministically based on the highest Q-value. In fact, it is directly estimating the optimal action-value function q_*. There are different options for the behaviour policy, discussed in the next section. On the contrary to off-policy learning, there are *on-policy* methods that estimate the value of a policy while at the same time using it for control.

3.1.6 Exploration

In order to learn how to act optimally, one cannot only *exploit* the current knowledge by taking greedy actions that are estimated to give the most future return. It is also important to *explore* by taking non-greedy actions. In the case where an estimate, Q, of the action-value function is available, the greedy action is given by $\arg\max_a Q(s, a)$ for a certain state $S_t = s$.

The simplest and by far most common way of exploring is by using an ϵ-*greedy* policy. With this policy, the greedy action is chosen with probability $1-\epsilon$ and a random action is taken with probability ϵ, where $\epsilon \in [0, 1]$ is referred to as the *exploration rate*. It is common to anneal the exploration rate from a value close to 1, in the beginning of the learning phase, to some value close to 0 in the end.

Another way of doing exploration when the Q function is available is to construct a Boltzmann distribution over all actions and sample from it. This is called *Boltzmann exploration*. In order to control the amount of exploration, a temperature parameter, τ, is used. This parameter is often also annealed in a similar way as the exploration rate. The distribution is defined as follows:

$$\Pr\{A_t = a | S_t = s\} = \frac{e^{Q(s,a)/\tau}}{\sum_{b \in \mathcal{A}} e^{Q(s,b)/\tau}}. \tag{3.15}$$

Actions with higher Q-values have a higher probability to be sampled, thus the exploration is focused on the actions that are estimated to be the best once, which in principle should be beneficial. However, the authors of [27] claim that convergence of the learning algorithms can be slow unless the temperature parameter is tuned carefully.

3.2 Neural Networks

The term *neural network* is loosely inspired by attempts to represent biological information processing in a mathematical fashion. Such first attempts started as early as 1943 [28] and the big breakthrough came with the *backpropagation* algorithm in the mid 80's. It enabled efficient training of neural networks and generally learning of parameters in computational graphs [29]. Today, neural networks are almost synonymous with machine learning and in fact they play an important part in many fields such as computer vision, speech recognition, and natural language processing [30].

3.2.1 Multilayer Perceptron

What neural networks do is to approximate functions. The simplest architecture is the multilayer perceptron, sometimes called the feed-forward network, which uses a flexible parametric form for the approximation. Assuming we want to approximate the mapping $\boldsymbol{y} = g(\boldsymbol{x}),\ g : \mathbb{R}^m \mapsto \mathbb{R}^n$ with some other function f, the network uses a number, l, of linear transformations followed by nonlinearities, $h(\cdot)$ called *activation functions*. The network takes $\boldsymbol{x}$ as input and outputs $\boldsymbol{y}^*$ which is some approximation of $\boldsymbol{y}$:

$$\begin{aligned} \boldsymbol{x}_1 &= h_1(\boldsymbol{w}_1^T \boldsymbol{x} + \boldsymbol{b}_1) \\ \boldsymbol{x}_2 &= h_2(\boldsymbol{w}_2^T \boldsymbol{x}_1 + \boldsymbol{b}_2) \\ &\vdots \\ \boldsymbol{y}^* &= h_l(\boldsymbol{w}_l^T \boldsymbol{x}_{l-1} + \boldsymbol{b}_l) \end{aligned} \tag{3.16}$$

This recursive stack of linear and nonlinear functions is referred to simply as the "neural network" and the number l is called the number of layers. Due to the free parameters, $\theta = (\boldsymbol{w}_1, \boldsymbol{b}_1, \ldots, \boldsymbol{w}_l, \boldsymbol{b}_l)$, this kind of network can in theory approximate any continuous function and is therefore said to be a *universal function approximator* [3].

3.2.2 Activation functions

Generally, different activation functions are used in the last layer of the network and in the intermediate layers. In the last layer, the activation

function is adjusted to the task. For regression tasks, when the output is a scalar, there is a linear activation so that any real value can be attained. Denote the linear part of the last layer with $\boldsymbol{z} = \boldsymbol{w}_l^T \boldsymbol{x}_{l-1} + \boldsymbol{b}_l$. For a linear activation function, we have

$$\text{linear}(\boldsymbol{z})_i = z_i, \quad \text{for } i = 1, \ldots, K. \tag{3.17}$$

Instead, if the output is to represent probabilities for different classes, as in a classification tasks, a softmax activation is used:

$$\text{softmax}(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \quad \text{for } i = 1, \ldots, K. \tag{3.18}$$

In the intermediate layers, the most commonly used activation function is the Rectified Linear Unit (ReLu) [31]:

$$\text{ReLu}(\boldsymbol{z})_i = \max(0, z_i), \quad \text{for } i = 1, \ldots, K, \tag{3.19}$$

which gives a gradient of 1 for $\boldsymbol{z}_i > 0$, and 0 otherwise. Another variation is the leaky ReLu, which allows a small gradient even for the case of $\boldsymbol{z}_i \leq 0$ [32]:

$$\text{LeakyReLu}(\boldsymbol{z})_i = \begin{cases} z_i & \text{if } z_i > 0 \\ 0.01 z_i & \text{else} \end{cases}. \tag{3.20}$$

3.2.3 Loss Functions and Optimizers

In order to find good parameters θ, i.e. to train the network, a loss function $\mathcal{L}(\theta)$ is defined. In the case of regression, this can be the mean squared error of all training examples or in the case of classification a cross-entropy loss can be used [3]. This loss is then minimized using some variation of stochastic gradient descent (SGD) where an estimate of the gradient of the loss, $\nabla_\theta \mathcal{L}(\theta)$, is used to do parameter updates. In the SGD optimizer, the updates are done as follows:

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(\theta), \tag{3.21}$$

where η is the learning rate.

Another optimizer, which often provides faster convergence and better results, is Adam [33]. This optimizer computes adaptive learning rates for different components of the parameter vector, by using estimates of first and second moments of the gradients. In this case, the

update is more complicated than in equation (3.21), and the details can be found in [33]. There are a couple of advantages over SGD that are worth mentioning. First of all, because of the adaptive learning rates in the different components, the optimizer essentially establishes a trust region around the current parameter value. This also means that finding the correct hyperparameters becomes easier. Secondly, the optimizer has the property that it shrinks the adaptive learning rates when there is a lot of uncertainty of the gradient (when the second moment becomes large): a kind of automatic annealing. Otherwise, this is often done separately from the optimizer and requires even further hyperparameters to tune. Even though learning rates are adaptive, there is still a parameter in Adam called "learning rate", which approximately is an upper bound on the adaptive ones.

3.3 Deep Reinforcement Learning

Neural networks can be used as function approximators for either value functions, or directly for the policy of the agent. This kind of approach is generally called deep reinforcement learning.

3.3.1 Deep Q-Networks

In the case of Q-learning, section 3.1.5, the estimates of the optimal action-values, q_*, are stored in a table, Q and are updated for each new experience tuple that the agent collects. One issue is that if the state space is very large or continuous, this approach is simply not feasible due to memory constraints. Another problem with a large state space is that it will be very hard to visit, and thus learn the value of, all state-action pairs in the table; called *the curse of dimensionality* [8]. The idea in Deep Q-Networks (DQN) is instead to approximate q_* with a parametric function, Q_θ, specifically a deep neural network with parameter vector θ. In this way, all action-values are instead stored implicitly in the parameters of the network, where the number of parameters can be less than the number of different action-values. Furthermore, one of the great strengths of neural networks is their ability to generalize, which means that values of state-action pairs that have not been encountered can still have good estimates.

In order to make training of the network stable, the authors of DQN propose two ideas: *experience replay* and the idea of a *target network* [12].

Experience replay helps to remove correlation between experience tuples by storing them in a buffer, $\mathcal{D}$, sometimes also called a replay memory. Instead of doing parameter updates sequentially with experience tuples as they are collected, random batches of tuples are sampled uniformly from the buffer to do updates. This can be seen in equation (3.22) where the expectation is taken over experience tuples sampled uniformly from $\mathcal{D}$.

In standard Q-learning, $Q(S_t, A_t)$ is updated towards a "Q-target" consisting of $R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$, where Q is used to approximate the tail of the return by bootstrapping. When the action-value function depends on θ, which is constantly updated during training, the target is not stationary. This is the reason for the target network. It is a second neural network that is used to compute the Q-targets with parameters θ^-, which are periodically updated to θ. This will make the Q-targets closer to stationary, which in the original paper is empirically proven to greatly increase stability.

The DQN update uses a loss function for the neural network similar to the update of standard Q-learning, here instead trying to minimize the squared difference of the Q-target and the predicted Q:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim U(\mathcal{D})}\left[\left(r + \gamma \max_{a'} Q_{\theta^-}(s', a') - Q_\theta(s, a)\right)^2\right]. \tag{3.22}$$

3.3.2 Double Q-Learning and Double DQN

Q-learning is known to sometimes learn unrealistically high action-values because of the maximization step in the update. This max operation favours overestimated actions over underestimated ones, in some cases negatively affecting performance and skewing exploration. A proposed solution to this problem, for the tabular setting, is *Double Q-learning* [34], a method to reduce the overestimation. Here, two estimates Q_A and Q_B of the action-value function are kept to learn from the same MDP, but from different experiences so that they are different and unbiased. The updates for Q_A are made as follows:

$$\begin{aligned} Q_A(s, a) &\leftarrow Q_A(s, a) + \alpha\left[r + \gamma Q_B(s', a') - Q_A(s, a)\right], \\ a' &= \arg\max_b Q_A(s', b). \end{aligned} \tag{3.23}$$

Here the target action is chosen by estimator A, but the actual value is computed by estimator B.

This idea, of using two estimators, has also been utilized with success in the case of DQN, called *Double DQN* (DDQN) [14]. In DQN, there are two estimators readily available: the "online" network serving the role of estimator A and the target network as estimator B. In DDQN, the only difference from DQN is the loss function, which is now changed to:

$$\begin{aligned} \mathcal{L}(\theta) &= \mathbb{E}_{(s,a,r,s')\sim U(\mathcal{D})}\left[(r + \gamma Q_{\theta^-}(s', a') - Q_\theta(s, a))^2\right], \\ a' &= \arg\max_b Q_\theta(s', b). \end{aligned} \tag{3.24}$$

Before, the action in the Q-target was taken with respect to the target network. Now, this is taken with respect to the online network to decouple selection and evaluation of the max action.

3.3.3 Multiple Goals & Universal Value Function Approximators

In certain tasks, there are multiple goals $g \in \mathcal{G}$, where $\mathcal{G}$ is the space of possible goals, that we may try to achieve. They can for example be certain states of the environment. In this setup, the reward function is conditioned on the goal $r_g : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$. In the beginning of each episode, a state-goal pair is sampled from some distribution $p(s_0, g)$, then the goal is kept fixed until the end of the episode. In order for the agent to act in accordance with the goal, the policy is conditioned on it, $\pi : \mathcal{S} \times \mathcal{G} \mapsto \mathcal{A}$, so that in each time step the agent takes both state and goal as input. Similarly, the value functions have to be conditioned on the goal.

In [35], it is shown that it is possible to directly train an approximator to the Q-function when both state and goal are given as input. The approximator, referred to as a Universal Value Function Approximator, can generalize over both state-action pairs and goals to new unseen scenarios. Several variation of this idea was proposed in the same paper. One is the concatenated architecture where state and goal are simply concatenate, then given as input to the policy. The two-stream architecture is another more sophisticated solution. Here, the state and goal are encoded using separate neural networks ϕ and ψ, then the dot product, h, of those are taken and the result is then given as input. A third architecture is a two-stream architecture where the encoding is done

by using matrix factorization as targets for the encoding vectors, then again these vectors are combined using a dot product. These different architectures can be seen in Figure 3.2.

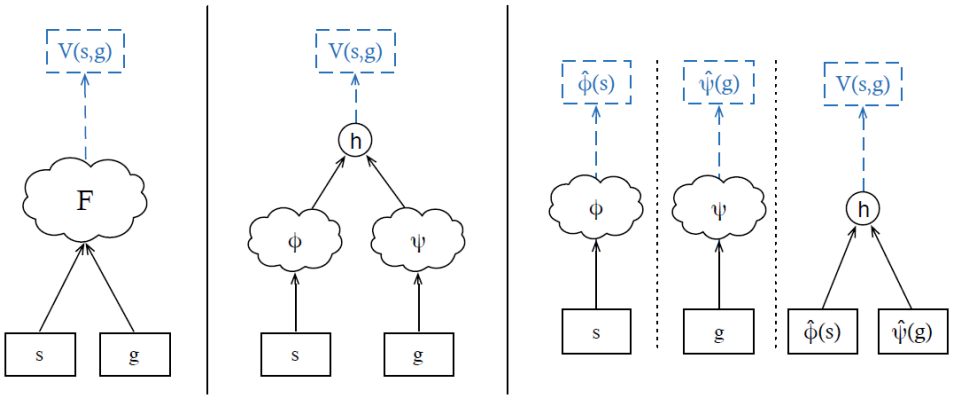

Figure 3.2: Universal value function approximation architectures. **Left**: Concatenated architecture. **Center**: Two-stream architecture with dot product. **Right**: Two-stream architecture with matrix factorization, followed by a dot product. Figure taken from [35].

3.3.4 Hindsight Experience Replay

One issue with the approach in section 3.3.3 is that varying goals, sampled from $\mathcal{G}$ in each episode, might rarely be reached. This will lead to sparse rewards and in turn to a slow learning process. A successful idea to circumvent this problem is *Hindsight Experience Replay* (HER) [19]. With this technique, after the termination of each episode, in hindsight, it is replayed with a state s that was actually reached as the goal instead of the original goal g of the episode. In this way, there are many more experiences where the reward for reaching the goal is provided. Thus, we might not learn how to achieve g, but we learn something about how to achieve s if that would have been the original goal and that is equally valuable since the goals vary. The authors also show that even if there is only one goal to be reached, this type of modified experience replay can still speed up learning and in some cases also lead to better performance.

There are some restrictions that need to be taken into consideration in order to use HER. First of all, an off-policy method has to be used so that a replay buffer $\mathcal{D}$, described in section 3.3.1, can be utilized, for ex-

ample DQN [12] or deep deterministic policy gradients (DDPG) used for environments with continuous action spaces [36]. This is so that replayed experiences can be put in the buffer to be sampled from during learning. Also, in order to use this method, the reward function has to be available so that rewards for the modified goals can be computed.

The authors of HER propose some different strategies for how to choose which states to set as goals in the hindsight replay. The simplest approach of setting the final state of the episode as the goal does not give the best performance. Instead, they find that a strategy they call *future* works best: for each transition in the episode, pick k random states that are reached after the current state in the episode and put those as the artificial goals. Then, compute the reward for that particular transition and input the experience tuples into the replay buffer. The variable k controls the ratio of experiences coming from hindsight and standard experience replay. For example, if $k = 2$ there are twice as many experiences coming from hindsight compared to standard replay. This way of populating the memory in the replay buffer is summarized in algorithm 1. Here, the concatenated architecture from section 3.3.3 is used to combine state and goal, where the symbol $\|$ denotes concatenation.

Algorithm 1: Populate Memory

Input:

- Episode trajectory $s_0, a_0, r_1, s_1, a_1, r_2, \ldots, r_T, s_T$
- replay buffer $\mathcal{D}$,
- reward function $r_g : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$.

$T \leftarrow$ trajectory length
for $t = 0$ **to** *T-1* **do**
  Retrieve $(s_t, a_t, r_{t+1}, s_{t+1})$ from trajectory
  Store $(s_t \parallel g, a_t, r_{t+1}, s_{t+1} \parallel g)$ in $\mathcal{D}$ `// Experience replay`
  for $i = 1$ **to** k **do**
    Sample time t' uniformly from $[t, T-1]$
    Set new goal g' to $s_{t'+1}$ `// Future strategy`
    Compute new reward $r' = r_{g'}(s_{t+1})$
    Store $(s_t \parallel g', a_t, r', s_{t+1} \parallel g')$ in $\mathcal{D}$ `// HER`
  end
end

3.4 A Measure of Radio Quality

One key metric for measuring the quality of a wireless connection in a cellular network is the *signal-to-interference-plus-noise ratio* (SINR). It summarizes the received power from the serving cell in the network and the interference coming from all other cells in the network into a single metric. In a position $\boldsymbol{x}$ it is defined by

$$\text{SINR}(\boldsymbol{x}) = \frac{P_i(\boldsymbol{x})}{N + \sum_{j \neq i} P_j(\boldsymbol{x})} \tag{3.25}$$

where cell i is the serving cell and N is a noise term that is often considered to be constant. The received power, $P_i(\boldsymbol{x})$, is dependent on position since there is attenuation due to path loss and shadow fading. The sum in the denominator is interference coming from the cells that are transmitting on the same part of the spectrum as the serving cell. Importantly, it shall be noted that the SINR is not stationary, but a random variable with some distribution that can be modeled using, for example, stochastic geometry [37]. Most often, the SINR is expressed in dB:

$$\text{SINR}_{dB}(\boldsymbol{x}) = 10 \log_{10} \left[\text{SINR}(\boldsymbol{x})\right]. \tag{3.26}$$

In the rest of the thesis, units of dB are used when referring to SINR.

Chapter 4

Method

4.1 Problem Definition

The focus of this work is applying reinforcement learning algorithms to the problem of controlling a drone in a cellular network. The control is high level in the sense that actions are directions to go towards, neither voltage nor torque to the motors. The use of reinforcement learning is motivated by the complex dynamics of the problem, if it is to be solved using optimization techniques there is a need to have the distribution of the radio quality (SINR). An internal representation of the distribution will instead be inferred through interaction by the agent. The study will be conducted using data from a network simulator issued by Ericsson Research. The simulator provides, in each position in space, a measure of the radio quality together with other information such as the location of the base stations and other users, if present. The propagation model follows specification 36.777 [38], developed for drones, from the 3rd Generation Partnership Project (3GPP), which is a standards organization developing protocols for mobile telephony. The propagation model, or channel model, includes path loss and shadowing effects and is discussed in some more detail in section 4.2.1.

However, the task is to have an agent learn to act optimally without access to this model, as would be the case in the real scenario. The assumption is that if a good control policy can be found in this environment, it can also be found in the real scenario. For the algorithm to be feasible for real deployment, it is of interest to have an efficient learning process that does not require a large amount of interaction.

4.2 Experimental setup

In this section, an overview of the experimental setup is described. The details of the studied cases are laid out and also how the state space, action space and reward function of the environment are chosen.

4.2.1 Environment

In order to evaluate the proposed method, a reinforcement learning environment was implemented. It was developed using data from Ericsson's network simulator that provided measures of radio quality in an area surrounding one base station and with interference coming from six other base stations, or sites, in a close proximity. This was chosen since the sites are positioned in a hexagonal pattern, seen in figure 4.1, so that only the closest ones are assumed to contribute with significant interference in the downlink.

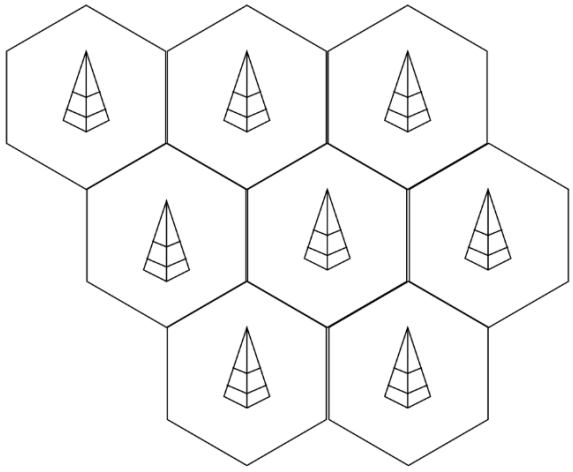

Figure 4.1: Hexagonal cellular architecture. Each site with six direct neighbours.

A square area surrounding the "middle" base station was chosen to represent the operational area of the drone, constraining it to pursue tasks within this region. The area can have any geometry, but was chosen as square for simplicity of implementation in this work. The downlink SINR measurements from the network simulator, in an area of 400×400 m^2 on an altitude of 30 m, can be seen in Figure 4.2, with one base station located in the origin and the other six are just outside

the bounds, but still providing interference. In the radio network simulator, each base station has three antennas, directed with 120° offset, mounted 25 meters above ground level. The three antennas are the reason for the beam shaped areas with higher SINR values. Some additional deployment parameters can be seen in Table 4.1.

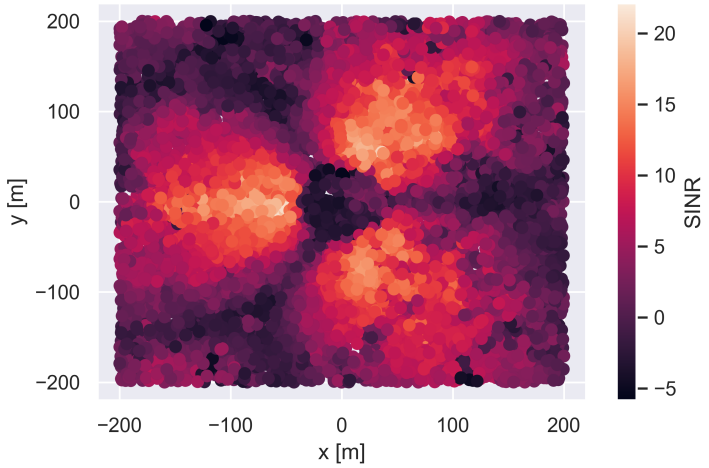

Figure 4.2: Measurements of SINR, in decibel, on 30 meters of altitude in an area surrounding a base station, located in the origin.

Parameter	**Value**
Sites	7
Base station power	48 dBm
Distance between sites	500 m
Carrier frequency	2 GHz
Carrier bandwidth	10 MHz
Propagation model	Spatial Channel Model 5G (3GPP TR-36.777) [38]

Table 4.1: Deployment parameters

The environment was implemented as a grid world, meaning that positions in the area were discretized into bins to reduce the size of the state space. The value of the SINR in each bin is taken as the average of the measurements, collected with the network simulator, that are located within the bounds of it. When the agent takes a step in a certain bin, the SINR value can be accessed with or without noise drawn

from $\mathcal{N}(0, \sigma^2)$. The noise is added to introduce variations in the radio quality rather than having a deterministic function depending only on location. This simulates a more realistic setting, in which there is always measurement noise.

Apart from using noise or not, it is possible to modify the environment further by using either two or three dimensions for the flight trajectories. In the experiments using two dimensions, the altitude was kept fixed to 30 meters and the grid was 20×20 bins, each with a size of 20×20 m^2. In the experiments using three dimensions, there is also an altitude range of $[20, 100]$ meters discretized into five bins with corresponding radio quality measurements for each altitude. Thus, in three dimensions there are $20 \times 20 \times 5$ bins of size $20 \times 20 \times 20$ m^3.

When a new episode is started, the environment samples a start and a goal position uniformly from all possible positions $\mathcal{P}$, following the description of multi-goal environments in section 3.3.3. The radio quality distribution is the same across all episodes, either completely static or with additional noise sampled from $\mathcal{N}(0, \sigma^2)$ in each time step. The only SINR value that the agent has access to during interaction with the environment is the value in the bin where the agent is currently located.

A visual representation of the environment can be seen in figure 4.3 where the SINR values in each bin are shown without noise for clarity.

4.2.2 Markov Decision Process

Formalizing the description of the environment above, in two dimensions we have the space of possible positions

$$\mathcal{P}_2 = \{(x_1, x_2) \in \mathbb{Z}^2 |\ x_1, x_2 \in [1, 20]\} \tag{4.1}$$

and in three dimensions the space is

$$\mathcal{P}_3 = \{(x_1, x_2, x_3) \in \mathbb{Z}^3 |\ x_1, x_2 \in [1, 20],\ x_3 \in [1, 5]\}. \tag{4.2}$$

After each interaction with the environment, an observation of the successor state (which in this case is the same as the successor state) and reward are returned. To limit the number of interactions an agent can take to reach the goal, episodes are terminated after a certain number of steps. For the experiments, it was chosen as ten times the longest possible path between two locations, i.e. 400 in two dimensions and 450 in three dimensions.

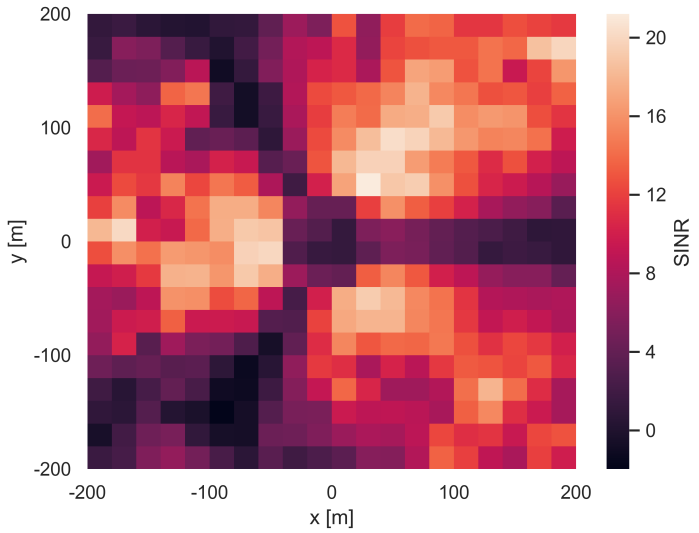

Figure 4.3: Visual representation of the two-dimensional environment, discretized into 20×20 bins.

State Space

The state space of the environment is given by $\mathcal{S} = \mathcal{P}_2$ or $\mathcal{S} = \mathcal{P}_3$ depending on the number of dimensions. The goals are also positions, $\mathcal{G} = \mathcal{P}_2$ or $\mathcal{G} = \mathcal{P}_3$. Since the SINR value, in the current position, is also available to the agent, it can be seen as included in the state. However, it is not input to the policy / Q-function, so we still use the convention $\mathcal{S} = \mathcal{P}$.

Action Space

The action space $\mathcal{A}$ is the set of directions to go towards, in two dimensions, there are actions to move north, south, east, and west. In three dimensions, there are also actions for changing altitude; up and down.

$$\mathcal{A} = \begin{cases} \{North,\ South,\ East,\ West\} & \text{if 2D} \\ \{North,\ South,\ East,\ West,\ Up,\ Down\} & \text{if 3D} \end{cases}. \tag{4.3}$$

Reward Function

At each time step, the agent executes an action and receives a reward based on the following reward function:

$$r_g(s') = \rho_{step} + \rho_{goal}\mathbb{1}\left[s' = g\right] + \rho_{sinr}\mathbb{1}\left[\text{SINR} < \Gamma\right] \tag{4.4}$$

where $\mathbb{1}\left[\cdot\right]$ is the indicator function, ρ_{step} is a step penalty, ρ_{goal} is a reward for achieving the goal, and ρ_{sinr} is a penalty that is given if the SINR value is below a certain threshold Γ. The values of the reward parameters can be seen in Table 4.2.

Parameter	Value	Description
ρ_{goal}	10	Reward for reaching the goal.
ρ_{sinr}	-1	Penalty for low radio quality, motivating the agent to avoid regions of low SINR.
ρ_{step}	-0.1	Step penalty, motivating the agent to move faster to the goal to not accumulate negative rewards.

Table 4.2: Table of parameters in the reward function.

These were found, empirically, to provide a balance between short paths and avoiding low radio quality areas. Since $\rho_{sinr} = 10\rho_{step}$, taking a path that is up to 10 times longer provides more cumulative reward (less penalty) than flying through an area of low SINR. Thus, the agent chooses to take detours around areas where it will be penalized for having a low radio quality. The reward parameters are not sensitive to the size of the area of operation. However, ρ_{sinr} and ρ_{step} are sensitive to the "depth" of the areas of low radio quality. If such regions require many steps to get through, and there is no way to fly around them, the agent can judge it better to wait out the end of the episode instead of flying through the low radio quality region and accumulate some penalties. This issue can be resolved by increasing the goal reward, but this also has other effects such as increasing the training time of the algorithm. If the goal reward is very large, all returns are in a very similar range, as long as the agent reaches the goal at all, meaning that most Q-values are updated to be roughly the same. This will make exploration slow due to the use of Boltzmann exploration. With this kind of exploration, actions are sampled from a distribution depending on the Q-values, if all Q-values are in a very close range the have the same probability of being sampled, loosing the effect of prioritizing actions with high estimated values which is the benefit of Boltzmann exploration.

Dynamics

To show that our reward function formulation is consistent with the MDP formalism, we can rewrite the dynamics in terms of the reward function and the transition function. Here the dynamics are also conditioned on the goal g,

$$\begin{aligned} p(s', r|s, a, g) &= p(r|s, a, g, s')p(s'|s, a, g) \\ &= p(r|g, s')p(s'|s, a) \\ &= r_g(s')\mathcal{T}(s'|s, a). \end{aligned} \tag{4.5}$$

The transition function $\mathcal{T}$ is deterministic in the testing environment. For example, if the *North* action is taken the drone will increment its second component of the position unless it is located on the upper border of the area of operation. In that case, it stays in the same position. The reward function, on the other hand, is stochastic because of the dependence of the SINR. The SINR distribution is assumed to be unknown, therefore we do not have access to the dynamics.

4.3 Learning Algorithm

The radio quality distribution is always the same in the area of operation, but the start and goal locations of the drone are always differing between episodes. They both belong to the set of possible positions $\mathcal{P}$ so in the beginning of each episode, s and g are sampled uniformly,

$$s_0,\, g \sim U(\mathcal{P}). \tag{4.6}$$

The start and goal positions can be sampled from another distribution than the uniform without affecting the learning algorithm. This setting is similar to the one described in section 3.3.3 where we want to learn to achieve multiple goals. The most successful approach for dealing with multiple goals is hindsight experience replay, which we adopt for our problem. We use DDQN as our learning algorithm, combining regular experience replay and hindsight experience replay with the future strategy as described in section 3.3.4. The experiences are collected using Boltzmann exploration, described in section 3.1.6, with the temperature annealed linearly over the episodes. When states and goals

are given as input to the DQN, and implicitly the policy, they are concatenated in a vector $s \parallel g$. We refer to our agent as DDQN-HER and the learning algorithm can be seen in Algorithm 2.

Algorithm 2: Learning, DDQN-HER

Input:

- Environment *env*,
- replay buffer $\mathcal{D}$,
- DQN weights θ,
- target DQN weights $\theta^- \leftarrow \theta$.

Populate $\mathcal{D}$ with M experiences collected using a random policy
for $episode = 0$ **to** N **do**
  $s_0, g \sim U(\mathcal{P})$
  for $t = 0$ **to** *T-1* **do**
    $a_t \sim \pi_\theta(s_t \parallel g)$ `// Boltzmann exploration`
    Take a_t and get r_{t+1}, s_{t+1} from *env*
  end
  Populate $\mathcal{D}$ with collected trajectory `// Algorithm 1`
  for $i = 1$ **to** opt_steps **do**
    Sample mini-batch B from $\mathcal{D}$
    Optimize θ with B using the loss in eq. (3.24)
  end
  Every C episodes, update target network: $\theta^- \leftarrow \theta$
end

The learning algorithm has a large number of hyperparameters. They were chosen by looking at previous work such as [12], [19], and by an informal hyperparameter search. All values can be seen in Table 4.3. Because of the larger state space in three dimensions compared to two dimensions, a larger number of episode and higher exploration fractions were used. Before learning occurs, the replay memory is populated, using Algorithm 1, with a number M of experiences that are collected using a completely random policy. This is a trick that is often utilized when using variations of DQN, for example in the two papers cited above, to get a diverse set of experiences in the replay memory right form the start of the learning phase.

Hyperparameter	Value	Description
mini-batch size	32	The number of experiences that are sampled from the replay buffer for each gradient descent update.
replay memory size	500000	The number of most recent experiences that are stored in the replay buffer.
number of episodes, N	30000, 50000	The total number of episodes the learning algorithm is run for. For 2D, 30000 was used and for 3D, 50000 was used.
target network update frequency, C	10	The frequency (measured in number of episodes) of the target network updates.
discount factor, γ	0.995	The discount factor used in the Double Q-learning update.
learning rate, η	0.001	The learning rate for the Adam optimizer, the momentum values were taken as the defaults in PyTorch [39].
initial temperature	2.0	The starting temperature τ of the Boltzmann distribution.
final temperature	0.1	The final temperature τ of the Boltzmann distribution.
exploration fraction	0.5, 0.7	The fraction of episodes that the temperature is reduced over. For 2D, 0.5 was used and for 3D, 0.7 was used.
replay start size, M	50000	The number of experiences collected with a random policy to populate the replay memory before learning.
optimization steps	30	Optimization steps to be taken after each episode.
hindsight ratio, k	2	The ratio of experiences coming from HER to the ones coming from standard experience replay.

Table 4.3: Table of hyperparameters for the learning algorithm.

The neural network, acting as an estimator of the optimal action-value function in the DDQN-HER, is a multilayer perceptron with 3 hidden layers of 50 units each. The hidden layers use the Leaky ReLu activation function, and the output layer uses a linear activation function. In the two dimensional case, this correspond to a total of 5554 parameters in the network and in the three dimensional case 5756 parameters or a model size of 44 and 46 kB respectively when the parameters are stored with double precision. The input to the network, the state, was scaled to the range $[0, 1]$ by dividing each dimension by the number of bins to provide more stable learning.

4.4 Evaluation

To evaluate the proposed method, 10000 episodes are run using a greedy policy after training has completed. This can be seen as estimating statistics of the different metrics, described below, using Monte-Carlo simulation. The setting is identical to the learning phase; the signal distribution is the same, start and goal positions are sampled uniformly at the beginning of each episode, and episodes are terminated if the drone has not reached the goal within a certain number of steps. Both the proposed DDQN-HER agent and the baseline are acting in the same instances of the environment for a fair comparison. The metrics are presented in the form of empirical cumulative distribution functions (CDFs). This is so that key values such as the median and different percentiles can be read directly from the diagrams. These are shown in the results chapter together with some summary key values.

4.4.1 Baseline Agent

The baseline agent is taking actions greedliy with respect to the goal, without regard to the radio quality. Thus, it will always take the shortest possible path by definition. It favours actions along the dimension where the distance between the current location and the goal is the furthest.

4.4.2 Metrics

To evaluate the algorithm, we look at a few different metrics. Ultimately, the objective is for the drone to spend as little time as possible in

areas of low radio quality. We define these areas to be where the SINR is measured below a threshold of $\Gamma = 3$ dB. This is the same threshold that is used in the reward function so that meaningful feedback can be given when the drone enters such an area. The SINR is not defined in the 3GPP specifications, therefore there is no official classification of the range of values. However, most user equipment (UE) vendors still uses SINR since it quantifies the relationship between the radio frequency (RF) conditions and the throughput well. The table shown in figure 4.4 represents a good classification of RF conditions for some common measures used, including SINR. Because SINR is not standardized by 3GPP, different such tables exist, but mostly they are well aligned. The value of the threshold, $\Gamma = 3$ dB, is within the range of Mid Cell conditions meaning that it is an acceptable value. It was chosen to provide test cases with distinct areas of low radio quality for the simulation, while being in the lower end of the acceptable range.

		RSRP (dBm)	RSRQ (dB)	SINR (dB)
RF Conditions	Excellent	>=-80	>=-10	>=20
	Good	-80 to -90	-10 to -15	13 to 20
	Mid Cell	-90 to -100	-15 to -20	0 to 13
	Cell Edge	<=-100	<-20	<=0

Figure 4.4: Radio Quality conditions for different measurement quantities [40].

SINR

The downlink signal-to-interference-plus-noise-ratio (SINR), measured in dB and defined in equation (3.26), was chosen as the measure of radio quality in this work. Naturally, this value is also used as a metric and most important is that there is a low probability of going below the threshold Γ. This is more important than having a very high SINR, since that does not provide further benefits.

Path Length Ratio

The drone should not only avoid areas of low radio quality, it should also reach the goal position quickly. Preferably, it should take flight trajectories that are as short as possible. To quantify this, we take the

ratio of the path length and the shortest path for each episode. Thus, the optimal value of the ratio is 1 and would be equal to 2 for a path twice as long as the shortest possible one.

Return

During learning, by design, the agent gets better at optimizing the expected return. Thus, this is the standard metric to look at for reinforcement learning algorithms to monitor performance and learning speed.

The baseline agent will get a fairly large return in each episode, due to how the reward function is constructed. It will always reach the goal using the shortest path - meaning that it will not be given many step penalties. However, it might get penalized for taking steps in low radio quality areas.

Since the baseline has a high expected return by default, it is of interest to see that our agent at least performs better with respect to the return, or sum of rewards, than the baseline. If not, the proposed algorithm is not able to handle this particular environment well. Consequently, the return is also used as a metric.

Altitude

In the case of 3D, in which the altitude of the drone can vary, it is of interest to see how much time is spent on each altitude. Since the network is designed for terrestrial use, the radio quality is often much higher on lower altitudes. Therefore, the expected behaviour is that the drone is maintaining a low altitude for a large fraction of the time of flight. In the results, this metric is presented using a empirical probability mass function (PMF), simply to show what fraction of time is spent on each altitude.

Qualitative Assessment

Since we have access to the SINR distribution, we can visualize the probabilities of measuring below the threshold in each position. However, the assumption is that the agent does *not* have access to these probabilities. Examples of, so called, probability maps on an altitude of 30 meters can be seen below in figures 4.5 and 4.6.

The probabilities are plotted for environments with and without added noise respectively, where only the 2D case is shown here for sim-

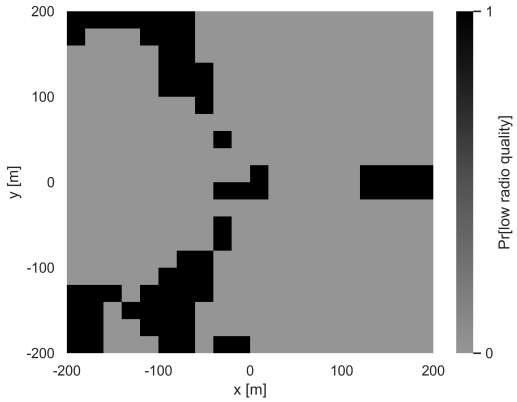

Figure 4.5: Probability map, without noise added to the SINR.

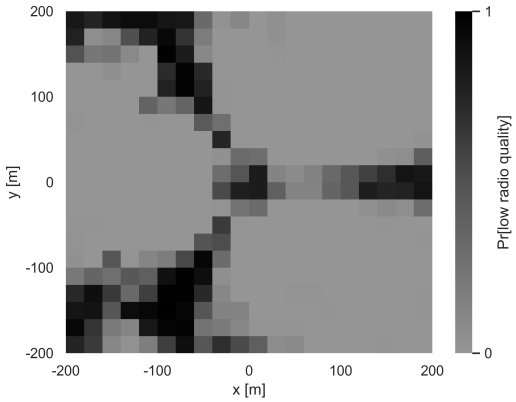

Figure 4.6: Probability map, with noise drawn from $\mathcal{N}(0, \sigma^2)$.

plicity. It is helpful to have these in order to visualize the policy and qualitatively evaluate the behavior of the agent. The resulting policies that we show in the results chapter are extracted using the greedy policy:

$$\pi_{greedy}(s \parallel g) = \arg\max_a Q_\theta(s \parallel g, a), \quad \forall\, s, g. \tag{4.7}$$

We can also visualize the state-value function by using the Q-network which gives access to the Q-values. First, we rewrite the state-value function from the definition in section 3.1.3 as follows:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi \left[G_t | S_t = s \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1} | S_t = s \right] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma v_\pi(s') \right], \end{aligned} \tag{4.8}$$

where in the second step, we used equation 3.4 and in the third step we expanded the expectation and used the fact that $\mathbb{E}_\pi \left[G_{t+1} | S_{t+1} = s' \right] = v_\pi(s')$. Technically, v_π should be conditioned on the goal, but this is omitted since the goal can be included in the state by letting $s \leftarrow s \parallel g$. In the same way, the action-value function can be rewritten to

$$q_\pi(s, a) = \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma v_\pi(s') \right]. \tag{4.9}$$

Combining these two equations gives us an expression for v_π in terms of q_π and π:

$$v_\pi(s) = \sum_a \pi(a|s) q_\pi(s, a), \tag{4.10}$$

If we want to visualize the estimated state-value function of the greedy policy we get

$$V_{\pi_{greedy}}(s \parallel g) = \max_a Q_\theta(s \parallel g, a), \quad \forall\, s, g. \tag{4.11}$$

which is used to produce the value function plots shown in chapter 5. These can give insight into the "thought process" of the agent, showing the estimated values of the states according to the current knowledge that exists implicitly in the weights of the neural network.

Chapter 5

Results & Analysis

5.1 Constant Altitude Without Noise

The first case is where all flight trajectories lie on an altitude of 30 meters and there is no noise in the SINR distribution, meaning that it is a deterministic function of position given by the data from the network simulator. A summary of the metrics are shown in Table 5.1. The reported SINR $< \Gamma$ is the ratio of measurements below the threshold to the total number of steps taken during all the evaluation instances. Resulting CDFs are shown in Figures 5.1 - 5.3. Visualizations of the policy and the value function for the proposed agent, DDQN-HER, are shown in Figures 5.4 - 5.7 and two different state-goal instances are displayed to show example of the behavior. One where the goal is within a region of low radio quality and another where the goal is in a region of high enough radio quality.

Agent	**SINR** $< \Gamma$	**Path ratio p90**	**Goal reached**
DDQN-HER	3.40%	2.00	99.82%
Baseline	13.48%	1.00	100%

Table 5.1: Comparison of DDQN-HER and greedy agent, constant altitude without noise.

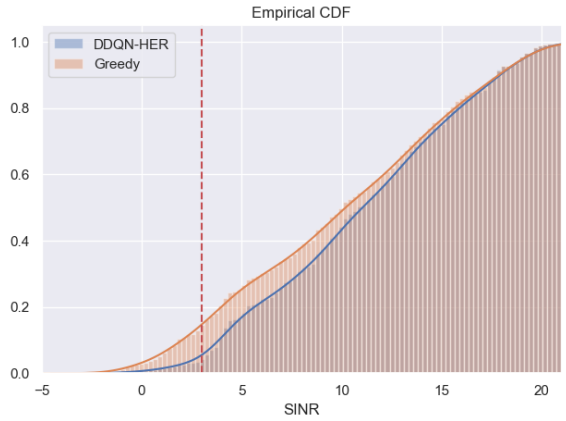

Figure 5.1: CDF for the SINR values. The red dashed line shows the SINR threshold.

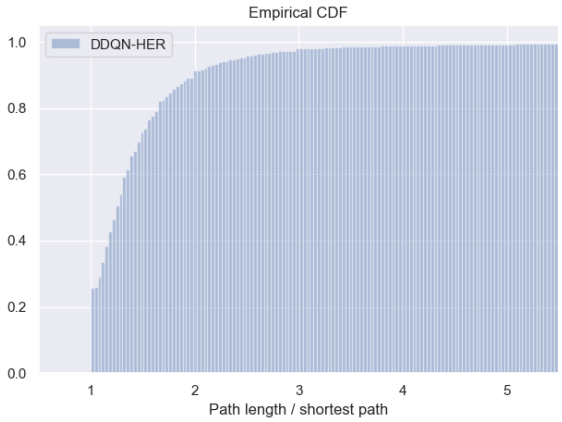

Figure 5.2: CDF for the path ratio.

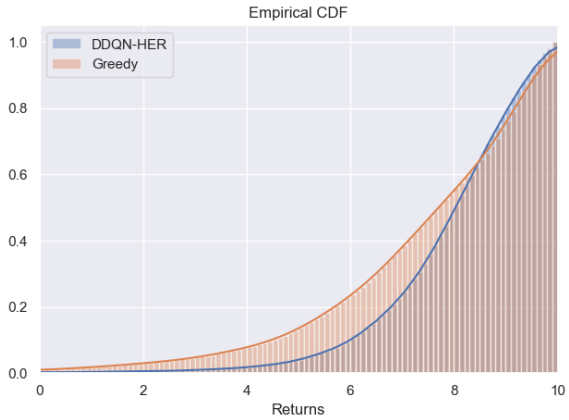

Figure 5.3: CDF for the return

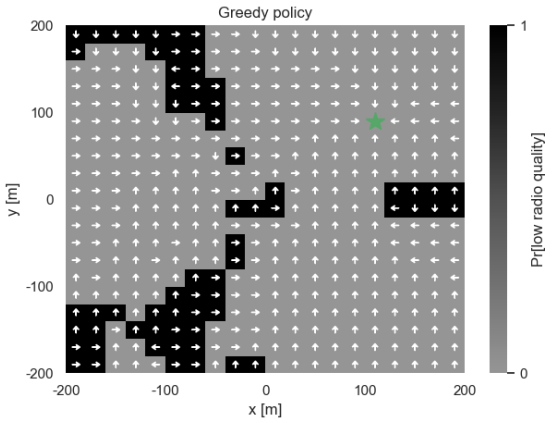

Figure 5.4: Policy visualization

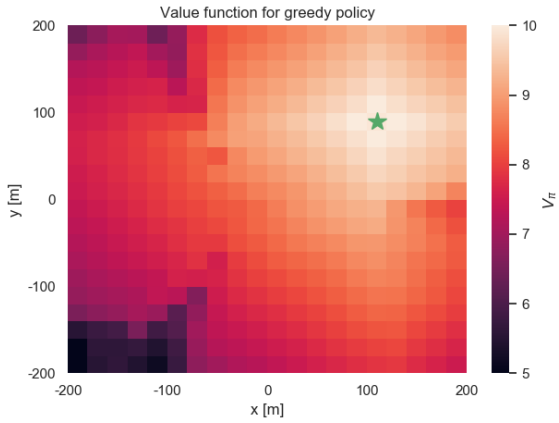

Figure 5.5: Value function

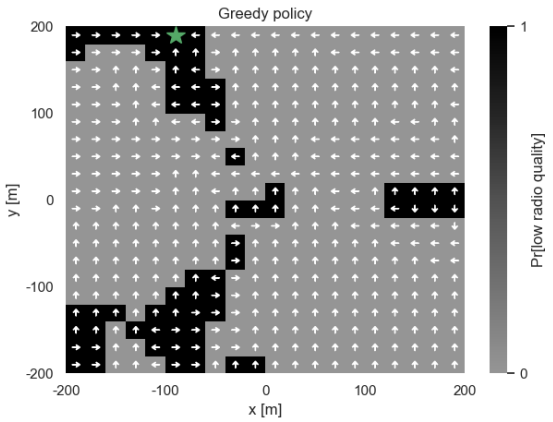

Figure 5.6: Policy visualization

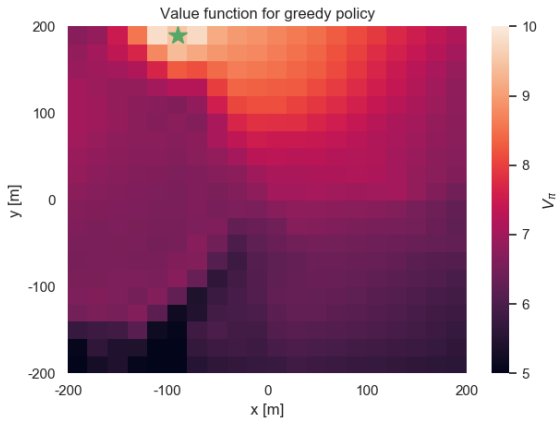

Figure 5.7: Value function

Analysis

The learning algorithm was run for $3 \cdot 10^4$ episodes, while there are $1.6 \cdot 10^5$ possible state-goal pairs. This means that all state-goal pairs have not been visited and the learning algorithm generalizes to unseen state-goal pairs well since the success rate of reaching the goal is 99.82%.

What is more interesting is to look at the SINR results. The percentage of total measurements below the SINR threshold can be interpreted as the probability of low radio quality in each step. The DDQN-HER agent has a probability of 0.0340 and the baseline 0.1348. Thus, that the probability of the proposed agent having low radio quality is roughly

one-fourth of the probability of the baseline, which is a significant improvement. Looking at the CDF of the SINR, Figure 5.1, it is evident that most of the probability mass is located above the threshold, shown with a red dashed line. This clearly shows that our agent learns to avoid the areas of low radio quality successfully. At the same time, the 90th percentile for the path ratio is 2.00, meaning that 90% of the paths are at most two times the length of the shortest possible path for the particular instance of start and goal position.

Figure 5.3 is also clearly showing an advantage for the DDQN-HER agent in terms of the return. There is barely any probability mass at the low returns, instead most of it is located at high returns where the goal is reached and not many penalties for low radio quality is given.

In the policy visualizations, the goal is marked with a green star and in each position, the action deemed optimal by the agent is plotted with an arrow. In the case of the value function, the goal is again marked with a green star and the color gradient in each position indicates the expected return being in that state. First of all, when we look at the policy visualizations for the two different instances, Figures 5.4 and 5.6, we see that the agent can reach goals both in and outside of areas with low radio quality. Secondly, it is evident that the trajectories the agent takes avoid areas of low radio quality, unless the goal is within such a region. In that case the agent accepts some SINR penalties in order to reach the goal position. For example, in Figure 5.4 trajectories going from the left half of the area toward the right side, where the goal is located, neatly bypass locations in the middle region that correspond to certain SINR penalties. This is the case in every instance, the agent takes detours around areas with SINR below the threshold, just as desired.

The value function plots, Figures 5.5 and 5.5, also give us some interesting insights. Positions close to the goal are always valued the highest and also the areas of low radio quality can easily be spotted (the darker spaces in the value plots). This is qualitative evidence that the agent learns to generalize to unseen goals and that it implicitly learns the signal distribution without having access to it.

5.2 Constant Altitude With Noise

The second case is where all flight trajectories lie on an altitude of 30 meters and the noise in the SINR distribution is drawn from $\mathcal{N}(0, \sigma^2)$

with standard deviation $\sigma = 2$ dB. A summary of the metrics are shown in Table 5.2 and resulting CDFs are shown in Figures 5.8 - 5.10. Visualizations of the policy and the value function for the proposed DDQN-HER are shown in Figures 5.11 - 5.14. Again, two different instances are shown. One instance where the goal is within a region of almost certain low radio quality and another where the goal is outside such a region.

Agent	**SINR** $< \Gamma$	**Path ratio p90**	**Goal reached**
DDQN-HER	5.87%	1.89	100%
Baseline	15.33%	1.00	100%

Table 5.2: Comparison of DDQN-HER and greedy agent, constant altitude with noise.

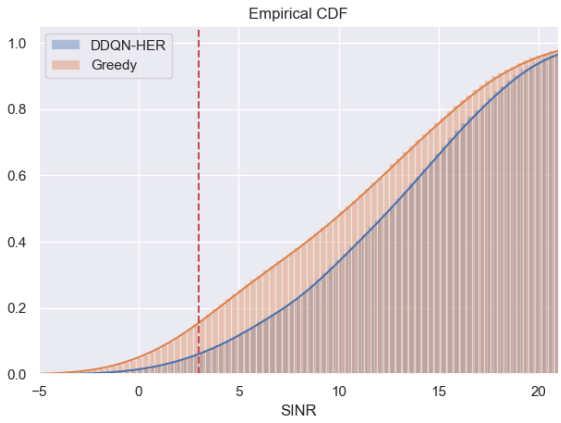

Figure 5.8: CDF for the SINR values. The red dashed line shows the SINR threshold.

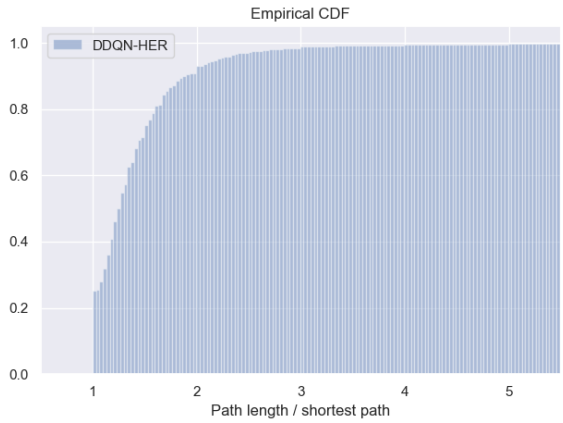

Figure 5.9: CDF for the path ratio.

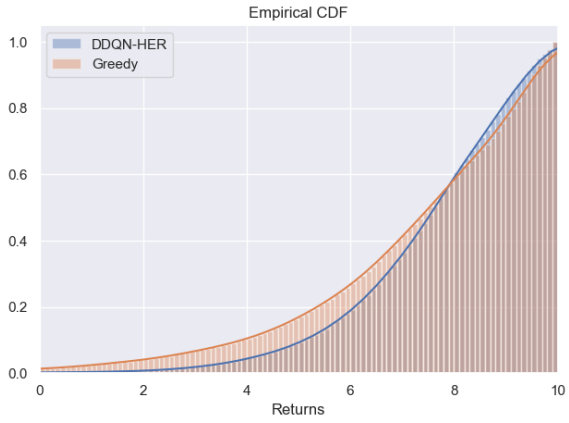

Figure 5.10: CDF for the return

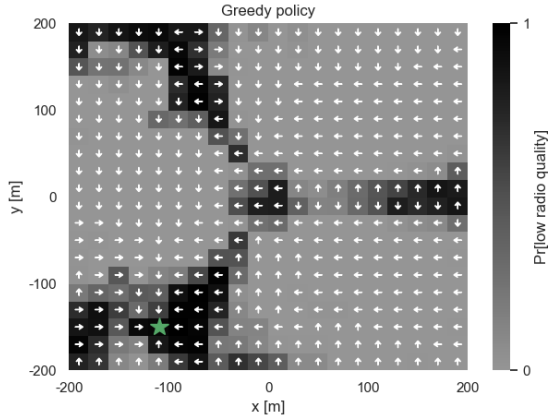

Figure 5.11: Policy visualization

Figure 5.12: Value function

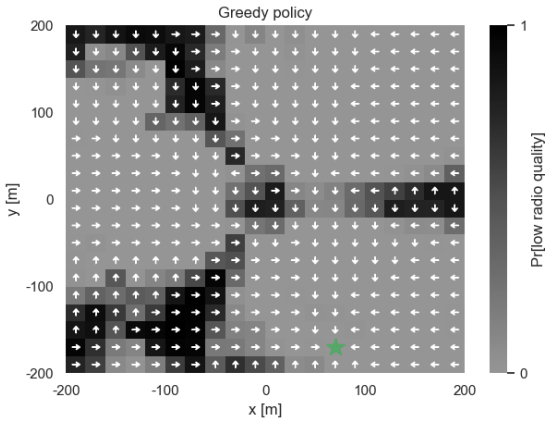

Figure 5.13: Policy visualization

Figure 5.14: Value function

Analysis

Analogous to the previous case, the learning algorithm is run for $3 \cdot 10^4$ episodes, while there are $1.6 \cdot 10^5$ possible state-goal pairs. Again, this means that all state-goal pairs have not been visited and the learning algorithm generalizes to unseen state-goal pairs very well since the success rate of reaching the goal now is at 100%.

The 90th percentile of the path ratio is reduced to 1.89%, indicating that 90% of the paths are shorter than 1.89 times the shortest possible paths. This means that the agent now chooses shorter flight paths compared to the previous case without measurement noise. Surprisingly, it seems that with measurement noise in the SINR, the agent moves with more confidence to the goals. Most likely, it is because the agent does not get penalized with the same certainty as in the case without measurement noise and exploits this fact.

In this scenario, with noise added to the SINR, the probability of going below the threshold is often somewhere between zero and one. This can be compared to the previous case without noise where the probability was either zero or one for a certain position. Instead, in this case, the area where the SINR can go below the threshold covers a larger portion of the area (see Figures 4.5 and 4.6). In the SINR results, we again see the same trend as in the previous setting without measurement noise. The proposed agent has a much lower probability of flying through an area of low radio quality. The probability is 0.0587 for the proposed agent and 0.1533 for the baseline. The probabilities

for both agents have increased compared to the case of no measurement noise, meaning that this task is more difficult. However, this was expected to be the case due to the dynamic SINR distribution. Still, the proposed DDQN-HER agent has a clear advantage over the baseline with respect to signal quality even after introducing measurement noise. This suggests that more complex environment dynamics can be handled well by the agent without modifications to the learning algorithm.

5.3 Varying Altitude With Noise

The third, most realistic, case is where flight trajectories may vary in altitude and the noise in the SINR distribution is drawn from $\mathcal{N}(0, \sigma^2)$ with standard deviation $\sigma = 2$ dB. The allowed altitudes for flying are $20, 40, 60, 80,$ and 100 meters. A summary of the most important metrics are shown in Table 5.3 and resulting CDFs are shown in Figures 5.15 - 5.17. In three dimensions it is harder to visualize the policy and value function, which is why they are omitted from the results in this case. In Figure 5.18, the empirical PMF for the altitude is shown for the proposed agent and the baseline to show the fraction of time spent on each altitude.

Agent	**SINR** $< \Gamma$	**Path ratio p90**	**Goal reached**
DDQN-HER	18.44%	2.11	100%
Baseline	75.21%	1.00	100%

Table 5.3: Comparison of DDQN-HER and greedy agent, varying altitude with noise.

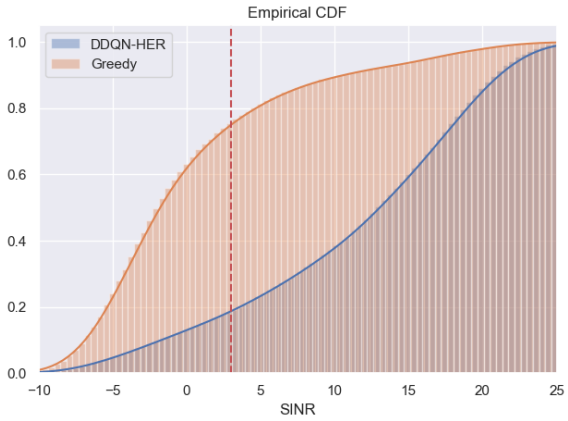

Figure 5.15: CDF for the SINR values. The red dashed line shows the SINR threshold.

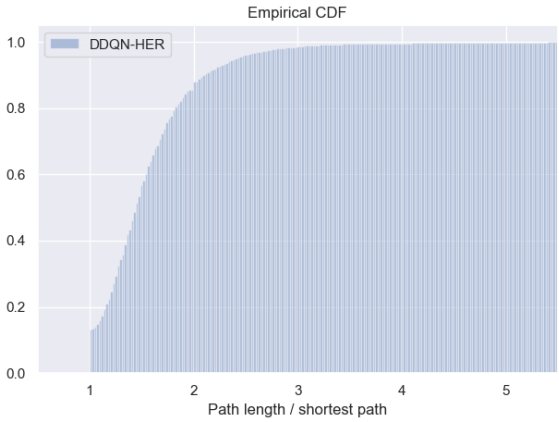

Figure 5.16: CDF for the path ratio.

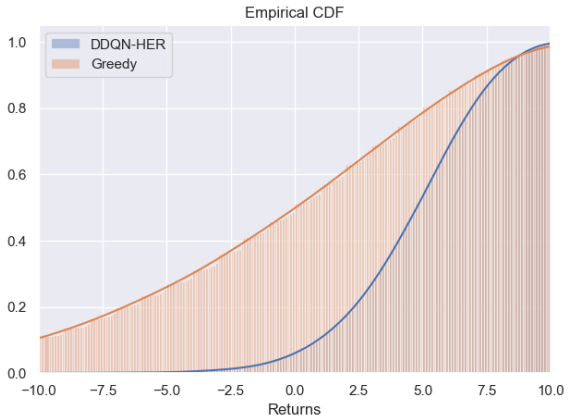

Figure 5.17: CDF for the return

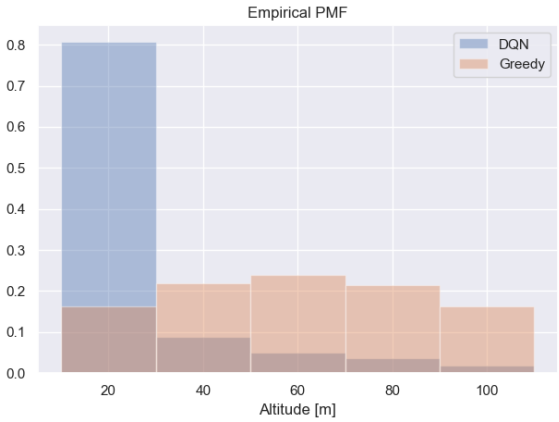

Figure 5.18: PMF for the altitude

Analysis

In this case, the learning algorithm is run slightly longer to compensate for the larger state space. It is run for $5 \cdot 10^4$ episodes and there are $8 \cdot 10^5$ possible state-goal pairs. Just as for the previous two cases, the learning algorithm generalizes to unseen state-goal pairs since it reaches the goals with 100% success rate. This means that there is no issue enlarging the state space and allowing for movement in 3D space. Also, it suggests that the state space can be made larger in other ways such as increasing the resolution of the grid, using a bigger area of operation,

or both while keeping the same learning algorithm.

The SINR results now look rather peculiar if compared directly with the previous two cases. The probability of measuring below threshold for DDQN-HER is now at 0.1844, see Table 5.3. What is reassuring is that the probability for the baseline is 0.7521, meaning that the environment is now a much harder task. The reason is that the signal data, coming from the radio network simulator, is very different on higher altitudes and most regions have SINR values below the defined threshold Γ. We already knew this would be the case, since the simulated network is designed for ground use and not for use on high altitudes.

If we look at the CDF of the SINR in Figure 5.15 for the baseline agent, we can get an idea of how the SINR really look like since it spends approximately equal amounts of time everywhere in the area of operation (remember that it flies the shortest path between completely random positions). More than 75% of the measurements are below the threshold and there is also a much wider range of SINR values, where a large portion of these are far below the threshold.

The good news is that the proposed agent still holds a clear advantage over the baseline, if longer paths are acceptable. The baseline has roughly four times as high probability of flying through areas of low radio quality and the DDQN-HER agent flies in trajectories that are at most 2.19 times the shortest paths in 90% of the instances. Most of the time, however, the paths are much shorter as can be seen in the path ratio CDF in Figure 5.16, for example the median trajectory length is around 1.45 times the shortest possible trajectory.

In this 3D scenario, it is difficult to visualize the policy and value function. Instead, we can look at the PMF of the altitude, Figure 5.18, to get an idea of what the DDQN-HER agent does to avoid areas of low radio quality. The vast majority of the flight time, about 80% is spent on the lowest altitude, where the radio quality is the highest. Thus, if the goal is on higher altitudes, the agent flies on the lowest altitude until it is below the goal, and then increments the altitude until it reaches the final position.

5.4 Ablation Study

To study the effects of the different components of the learning algorithm, an ablation study is performed where each is removed in turn.

In this ablation study, we only look at the return which quite well reflects how well the agent performs since the return is the sum of all rewards over an episode - providing a summary metric for path length, if the goal was reach, and if penalties are given for flying through zones of low radio quality. Learning curves are shown in Figure 5.19 for standard DQN and DQN with double learning (DDQN in figure), each with and without hindsight experience replay (HER). The experiment was repeated five times with different seeds (providing different initialization of the weights in the Q-estimator and different start-goal instances) and for each episode the return is averaged and reported in the figure together with one standard deviation.

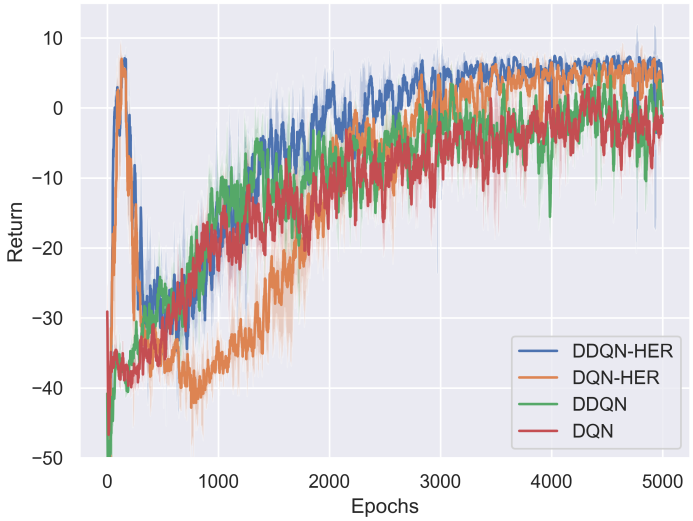

Figure 5.19: Study of the effects of different components of DDQN-HER.

Analysis

One noteworthy characteristic of the methods using HER is that the return "spikes" right in the beginning of the training phase. This happens because the agents learn to generalize to new goals almost immediately and therefore take straight paths to the goal. However, after some more training, where the agent receives SINR penalties, it learns to avoid such areas of low SINR and receives a higher return.

When double learning is applied without HER, it seems that it does

not have much effect, the red (DQN) and green curves (DDQN) are almost identical. However, when it is combined with HER it has the effect of speeding up the learning process. Most likely, it is because with double learning the estimates of the Q-values are not overestimated, leading to a more varied exploration.

Chapter 6

Discussion

6.1 Results Discussion

The different experiments in the previous chapter all show promising results for the proposed reinforcement learning agent. The first scenario, 2D trajectories without SINR measurement noise, display evidence that the agent learns to generalize to new goals. In a traditional control or planning algorithms this would never be an issue, but presents an obstacle for a solution based on reinforcement learning. The ablation study, see Figure 5.19, suggests that generalization is dealt with by introducing hindsight experience replay into the learning algorithm, without it the performance is reduced and learning is slower. In the first setting it also clear that the agent learns how to optimize for higher SINR value, by taking detours around areas of low radio quality. It can be seen quantitatively in the CDF of the SINR and qualitatively in the policy plots. The behavior is motivated by the constructed reward function, in particular the SINR penalty term, ρ_{sinr}. The length of the detours are affected by the relationship between the step penalty, ρ_{step} and the SINR penalty, ρ_{sinr} terms. These parameters are easy to interpret, the agent will chose to take detours that are at most ρ_{sinr}/ρ_{step} steps long and can be adjusted for a desired trade-off between path length and probability of having low radio quality.

In the second experimental setting, measurement noise is added to the signal distribution. This affects the performance of the agent slightly, but equally the performance of the baseline. Thus, the relative advantage of the proposed DDQN-HER agent over the baseline is still maintained, see Table 5.2. This proves that the proposed learning al-

gorithm can handle the uncertainty in the environment well, which is essential if the system is to be deployed in the real world.

The third scenario introduces an additional spatial dimension, enlarging the state-goal space. While the numerical values in the results now look slightly odd compared to previous experiments due to the signal being a lot weaker on high altitudes, the conclusion is still that the proposed agent holds an advantage in terms of maintaining strong signal, see Table 5.3. This is promising, because it means that the state-goal space can be increased without affecting the relative increase in performance over the baseline. This also indicates that it will be possible to increase the size of the state space in different ways and introduce further variables that affect it.

6.2 Validity

In the following subsections, we discuss the validity of the results in terms of a construct, internal, and conclusion standpoint with inspiration from [41].

6.2.1 Construct Validity

The concept of construct validity of an assessment is a degree of how well something that is supposed to be measured, is actually measured [42]. In this work, the data used for experiments come from a simulator and not from real actual measurements. The radio network simulator has been used extensively at Ericsson, therefore we can be certain that the data comes from a valid source. With that being the case, we are indeed "measuring" what we are supposed to measure.

6.2.2 Internal Validity

Internal validity of a study is how well it can rule out other causes of effects, than the ones investigated and it can be thought of as the degree of bias in the evaluation. In the same way as for the construct validity, there can be no doubt that the study is internally valid. This is because in the radio network simulator, nothing is being altered just because it is observed as it would be in, for example, a social study. Therefore, we can conclude that the study is also internally valid.

6.2.3 Conclusion Validity

Conclusion validity asks if what we observe in the study can be trusted to actually prove a relationship of interest [43]. In this work, we wanted to show that reinforcement learning successfully can be applied to the task of controlling a drone in a cellular network while maintaining high radio quality. Since the work was done in simulation we could easily do test simulations showing that the DDQN-HER agent had an advantage over the baseline. With use of 10000 test simulations that supports the proposed advantage, there is a strong belief that the conclusion is valid and that reinforcement learning can be applied to the task.

6.3 Sustainability and Ethical Aspects

Generally, autonomous systems have the potential of changing our world to the better if they are approached with great care and responsibility. However, there are many things that can go wrong. One difficulty of these systems is that even if they are created out of good intent, they can often be used for adversarial purposes, in warfare and sometimes with unforeseen side effects. This needs to be taken into consideration both by the designer and the organizational body that is responsible for deployment.

In this work, the level of autonomy is at the present stage hard to exploit or use to cause any harm. Although, it has the capability of bringing applications with positive societal impact into life. For example, drone delivery will often provide a good alternative to delivery with cars or trucks, which in general are more hungry for energy. This is one example of how an autonomous drone system can provide advantage in terms of sustainability compared to the current option.

Another application is surveillance, which, if used with the right intent, can have impacting, positive environmental and societal benefits. There are numerous examples of important areas of utilization such as monitoring wild fires and areas of draught. An almost identical system can also be used to be used for search and rescue systems in hazard zones after earthquakes or other catastrophic scenarios.

This work is a starting point to build similar, self learning, autonomous systems with many plausible, positive, sustainable and societal benefits.

Chapter 7

Conclusion

7.1 Summary

The intention of this thesis was to see if a proof of concept solution, based on reinforcement learning, could be established for the task of controlling a drone in a cellular network. The experiments clearly suggests that reinforcement learning indeed is a viable approach and a specific method was proposed based on Double Deep Q-Networks and Hindsight Experience Replay. The method, referred to as DDQN-HER, has the advantage of generalizing to unseen scenarios and the decision making process can be visualized to gain insight into the "thoughts" of the agent. The latter feature is a great for sanity checks and can be used for qualitative evaluation of the behavior.

The suggested reward function is what motivates the agent to pursue the wanted task of getting to the goal while avoiding low SINR. It was constructed to be easy to interpret and tune for different trade-offs between trajectory length and probability of low radio quality. Furthermore, the reward function works equally well for a static radio signal as for one with measurement noise owing to the fact that the learning algorithm handles uncertainties well.

The initial hurdles of varying start and goal positions, together with an unknown radio quality distribution are thus overcome. Therefore, the conclusion of this work is that reinforcement learning might be a feasible solution to network drone control. However, there are simplifications in this work and since all experiments are conducted in simulation, it is hard to know how well the method will work in a real scenario.

The end goal of a future agent, similar to the one presented here, is to be trained in simulation and then deployed in the real world. The idea is that even though signal propagation is slightly incorrect in simulation, the agent can learn to perform reasonably well in the real case. Over time in deployment, the agent can adjust its behavior and instead optimize its flight trajectories with respect to the real signal distribution.

7.2 Future Work

One interesting direction for future work is to make the signal distribution dependent on external factors such as throughput of other UE's. The current stage of this work, where the distribution is completely stationary is a good starting point and a fair approximation, but it is not entirely realistic. For the method to be more useful, future works should include such external perturbations in the data. If the cause of the perturbations is clearly related to a variable that would be accessible for each individual UE, then that should also be included in the state-space so the information can be taken advantage of by the agent.

Another intriguing direction of future work is to restrict the flight trajectories to avoid certain areas. In a real setting, there will be buildings blocking parts of the airspace in the lower altitudes. Furthermore, there are entire regions where drone flight is completely forbidden, for example above airports. How to deal with these restrictions is something that can be investigated in future works. One simple idea is to end episodes when these areas are flown into and to give the agent a large penalty (negative reward) when it happens.

Bibliography

[1] 3GPP. *Specification 21.915. Release 15*. Tech. rep. 3rd Generation Partnership Project (3GPP), 2017. URL: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3389.

[2] Goldman Sachs Research. *Drones: Reporting for Work*. 2017. URL: http://www.goldmansachs.com/our-thinking/technology-driving-innovation/drones/.

[3] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006. ISBN: 0387310738.

[4] Sergey Levine et al. "End-to-end training of deep visuomotor policies". In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1334–1373. ISSN: 1532-4435.

[5] David Silver et al. "Mastering the game of go without human knowledge". In: *Nature* 550.7676 (2017), p. 354. ISSN: 1476-4687.

[6] Kai Arulkumaran et al. "A brief survey of deep reinforcement learning". In: *arXiv preprint arXiv:1708.05866* (2017).

[7] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited, 2016.

[8] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018. ISBN: 0262352702.

[9] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*. Vol. 37. University of Cambridge, Department of Engineering Cambridge, England, 1994.

[10] Christopher J C H Watkins and Peter Dayan. "Q-learning". In: *Machine learning* 8.3-4 (1992), pp. 279–292. ISSN: 0885-6125.

[11] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. 1989.

[12] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), p. 529. ISSN: 1476-4687.

[13] Tom Schaul et al. "Prioritized experience replay". In: *arXiv preprint arXiv:1511.05952* (2015).

[14] Hado Van Hasselt, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-Learning." In: *AAAI*. Vol. 16. 2016, pp. 2094–2100.

[15] Ziyu Wang et al. "Dueling network architectures for deep reinforcement learning". In: *arXiv preprint arXiv:1511.06581* (2015).

[16] Matteo Hessel et al. "Rainbow: Combining improvements in deep reinforcement learning". In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[17] Richard S Sutton. "Dyna, an integrated architecture for learning, planning, and reacting". In: *ACM SIGART Bulletin* 2.4 (1991), pp. 160–163. ISSN: 0163-5719.

[18] Baolin Peng et al. "Deep dyna-q: Integrating planning for task-completion dialogue policy learning". In: *arXiv preprint arXiv:1801.06176* (2018).

[19] Marcin Andrychowicz et al. "Hindsight experience replay". In: *Advances in Neural Information Processing Systems*. 2017, pp. 5048–5058.

[20] Ronald J Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8.3-4 (1992), pp. 229–256. ISSN: 0885-6125.

[21] John Schulman et al. "Trust region policy optimization". In: *International Conference on Machine Learning*. 2015, pp. 1889–1897.

[22] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[23] Aviv Tamar et al. "Value iteration networks". In: *Advances in Neural Information Processing Systems*. 2016, pp. 2154–2162.

[24] Volodymyr Mnih et al. "Asynchronous methods for deep reinforcement learning". In: *International conference on machine learning*. 2016, pp. 1928–1937.

[25] John Schulman. *Optimizing expectations: From deep reinforcement learning to stochastic computation graphs*. 2016.

[26] Richard Bellman. "Dynamic programming". In: *Science* 153.3731 (1966), pp. 34–37. ISSN: 0036-8075.

[27] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. "Reinforcement learning: A survey". In: *Journal of artificial intelligence research* 4 (1996), pp. 237–285. ISSN: 1076-9757.

[28] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133. ISSN: 0007-4985.

[29] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning internal representations by error propagation*. Tech. rep. 1985.

[30] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), p. 436. ISSN: 1476-4687.

[31] Kaiming He et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[32] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. "Rectifier nonlinearities improve neural network acoustic models". In: *Proc. icml*. Vol. 30. 1. 2013, p. 3.

[33] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[34] Hado V Hasselt. "Double Q-learning". In: *Advances in Neural Information Processing Systems*. 2010, pp. 2613–2621.

[35] Tom Schaul et al. "Universal value function approximators". In: *International conference on machine learning*. 2015, pp. 1312–1320.

[36] Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[37] Ahmad AlAmmouri, Jeffrey G Andrews, and François Baccelli. "SINR and throughput of dense cellular networks with stretched exponential path loss". In: *IEEE Transactions on Wireless Communications* 17.2 (2018), pp. 1147–1160. ISSN: 1536-1276.

[38] 3GPP. *Specification 36.777. Enhanced LTE support for aerial vehicles*. Tech. rep. 3rd Generation Partnership Project (3GPP), 2017. URL: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3231.

[39] Adam Paszke et al. "Automatic differentiation in PyTorch". In: (2017).

[40] Arimas. *RSRQ To SINR Relation*. 2016. URL: https://arimas.com/164-rsrq-to-sinr/.

[41] Hamid Reza Faragardi. *Optimizing timing-critical cloud resources in a smart factory*. 2018.

[42] Lee J Cronbach and Paul E Meehl. "Construct validity in psychological tests." In: *Psychological bulletin* 52.4 (1955), p. 281. ISSN: 1939-1455.

[43] William M.K. Trochim. *Conclusion Validity*. 2006. URL: https://socialresearchmethods.net/kb/concval.php.