Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Telecommunications Engineering

Machine Learning-based Dynamic Resource Allocation for 5G Services

Master's thesis

Bc. Aida Madelkhanova

Study program: Electronics and Communications
Branch of study: Communication Systems and Networks
Supervisor: Doc. Ing. Zdeněk Bečvář, Ph.D.
Co-supervisors: Navid Nikaein, Thrasyvoulos Spyropoulos (EURECOM)

Prague, January 2020

Declaration

I hereby declare that I have completed this thesis on my own and that I have only used the cited sources. I have no objection to use of this work in compliance with the act "§60 Zákon č.121/2000 Sb." (copyright law), and with the rights connected with the copyright act including the changes in the act.

Prague, 6 January 2020

....
Bc.Aida Madelkhanova

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Madelkhanova Aida**	Personal ID number:	**434977**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Telecommunications Engineering**		
Study program:	**Electronics and Communications**		
Branch of study:	**Communication Systems and Networks**		

II. Master's thesis details

Master's thesis title in English:

Machine Learning-based Dynamic Resource Allocation for 5G Services

Master's thesis title in Czech:

Dynamická alokace prostředků založená na strojovém učení pro služby 5G sítí

Guidelines:

Study resource allocation in a software virtualized architecture with network slicing as a key flexibility enabler towards future service-oriented 5G networks. The resource allocation in such virtualized environment with slices is NP-hard problem, and thus it is important to find an optimal way to satisfy the resource requirement of all slices with even unpredictable traffic for some of them. The objective of this thesis is to investigate the allocation of multi-type resources (bandwidth, computing, storage, etc.) from the end-to-end perspective, spanning both the core network and radio access network where users reside and consume 5G services. Consider the real case where users and/or slices enter and leave the system at any time. Furthermore, investigate machine learning-type algorithms and improve or propose new machine learning-based algorithms for the dynamic allocations of multi-type resources with a dynamic arrival and departure of the users and slices. The proposed algorithm should be validated both with simulation and real data-set.

Bibliography / sources:

[1] M. Leconte, G. S. Paschos, P. Mertikopoulos, U. C. Kozat, 'A resource allocationframework for network slicing,' IEEE INFOCOM 2018, pp. 2177–2185, April 2018.
[2] G. Wang, G. Feng, W. Tan, S. Qin, R. Wen, S. Sun, 'Resource Allocation for Network Slices in 5G with Network Resource Pricing,' IEEE Global Communications Conference, December 2017.
[3] R. S. Sutton, A. G. Barto, 'Reinforcement learning - an Introduction,' Adaptive computation and machine learning, MIT Press, 1998.

Name and workplace of master's thesis supervisor:

doc. Ing. Zdeněk Bečvář, Ph.D., Department of Telecommunications Engineering, FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **05.08.2019** Deadline for master's thesis submission:

Assignment valid until: **19.02.2021**

doc. Ing. Zdeněk Bečvář, Ph.D. Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.

Date of assignment receipt

Student's signature

Abstrakt

Rychle rostoucí počet technologií páté generace 5G vyžaduje zkoumání nových strategií správy síťových prostředků. Datově řízené strojové učení umožňuje přístupy vyvinuté k dosažení výkonu s dostupnou výpočetní složitostí. Zpětnovazební učení je považováno pro budoucí inteligentní sítě. V této práci jsou uvažovány přístupy zpětnovazebního učení a hlubokého zpětnovazebního učení pro přidělování rádiových zdrojů v síti 5G. Jsou studovány čtyři algoritmy zpětnovazebního učení,na základě kterých se dále vyvíjí konkrétní design. Zanalyzována výkonnost každého vzdělávacího rámce a poskytováno porovnání (z hlediska průměrné odměny) mezi nimi. Hodnocení prokazuje obecnou použitelnost popsaných konceptů. Výsledky simulací ukazují, že techniky hlubokého zpětnovazebního učení předčí jednoduché Q učení.

Klíčová slova : 5G, Zpětnovazební Učení, Q-učení

Abstract

The rapidly increasing number of 5G technologies forces an investigation of new network resource management strategies. Recently, the data-driven Machine Learning (ML) enabled approaches developed to obtain optimal performance with affordable computational complexity. Reinforcement Learning (RL) is regarded for future intelligent networks. In this thesis RL and Deep Reinforcement Learning (DRL) approaches are considered for radio resource allocation in 5G network. We study four state-of-the-art RL algorithms, based on this, the concrete RL design is further developed. We analyze the performance of each learning framework and provide comparisons (in terms of the average reward) between them. The evaluation proves the general applicability of the described concepts. Simulations results show that DRL techniques outperform the simple Q learning.

Keywords: 5G, Reinforcement Learning, Q learning

List of Figures

List of Acronyms

5G	fifth generation. VIII, 1–3, 5, 17, 22, 43
AC	Actor-Critic. VIII, 4, 21, 35–38, 40–43
AI	Artificial Intelligence. 3
BS	base station. VIII, 20, 22
CSI	channel state information. 4
D-DQN	Double Deep Q Network. VIII, XII, 1, 4, 15, 21, 27, 28, 32–34, 37, 40–43
DL	Deep Learning. 1, 3, 43
DNN	Deep Neureal Network. 13, 35, 39, 43
DQL	Deep Q Learning. 3, 4, 12, 14
DQN	Deep Q Network. VIII, 1, 4, 14, 15, 21, 27–32, 40–43
DRL	Deep Reinforcement Learning. VII, VIII, 1–4, 13, 35, 36, 43
DRQN	Deep Recurrent Q-Network. 4
eMBB	Extreme Mobile Broadband. 5, 22
IID	independent and identically distributed. VIII, 23, 30, 33, 36, 40, 41, 43
IoT	Internet of Things. 6
LSTM	Long Short-Term Memory. 4
MDP	Markov Decision Process. 7, 10, 11, 13
ML	Machine Learning. VII, VIII, 1, 3, 7
mMTC	Massive Machine Type Communication. 5, 6, 22
NN	Neureal Network. VIII, 13, 21, 39, 43
NS	network slice. 1
NSP	Network Service Provider. 17
PER	packet error rate. 5, 6

QoS	Quality-of-Service. 1, 3–5
RAN	Radio Access Network. 5, 17
RB	resource block. VIII, 17–20, 22, 24, 37
RL	Reinforcement Learning. VII, VIII, 1–4, 7–11, 13, 20, 43
RNN	Recurrent Neural Network. 4, 43
SE	Spectrum Efficiency. 5
SL	supervised learning. 7, 8
SMDP	Semi-Markov Decision Process. 3
TD	Temporal Difference. 16
URLLC	Ultra-reliable and low-latency communications. 5, 6, 22
USL	unsupervised learning. 7
V2V	vehicle to vehicle. 3

Contents

1 Introduction

A whole new era of mobile communications, i.e., the 5G of mobile communication systems has been introduced to satisfy the explosive growth in capacity and coverage demands. The 5G mobile networks are designed to support a wide variety of innovative services, which require different needs in terms of latency, bandwidth, reliability, and flexibility [2]. Network resource management becomes an increasingly challenging step that requires proper design to advance network performance. In this situation, network slicing is a promising technology for 5G networks to provide services customized to the unique Quality-of-Service (QoS) specifications of users. Driven by the increased wireless data traffic from various application scenarios, efficient resource allocation schemes should be used to enhance the flexibility of network resource allocation and capacity of 5G networks based on network slicing. The aim of slicing is to allow simplicity and better use of network resources by providing only the required network resources to meet the specifications of the slices enabled in the system.

How to make efficient utilization of radio resources in 5G technologies is a brand-new challenge. Existing approaches used for previous generations are not capable to provide optimized resource allocation for reliable communication in 5G systems due to . In last few years some studies have shown an increased interest of integrating ML methodologies to learn the optimal network resource management strategies [3]. Since the accurate information and the complete model of the environments in mobile communications are unknown, the RL framework is a good match to solve optimization problems in wireless networks. Recently in the fields of RL promising results have been shown, owing much to developments in ML and Deep Learning (DL) [4, 5].

The key contributions of this thesis are summarized as follows:

— We thoroughly study different RL and DRL [6] techniques and investigate the use cases of DRL agents.

— We formulate the network slice (NS)s resource allocation problem and employ RL techniques to solve it. Four state of the art RL agents are trained and deployed to this model: Tabular Q-learning, DQN, D-DQN, Actor-Critic. Each technique is evaluated in terms of efficiency and feasibility. Simulations results show that DRL algorithms outperform the simple Q learning.

The structure of the thesis is as follows. The next chapter gives an overview of works related to the topic of this thesis. Chapter 3 focuses on the specifications of 5G networks and concludes the chapter by explaining the need for an integrated approach to effective network resource management. Then, Chapters 4 and 5 describes on the theory behind RL and DRL in detail. We focus on applied algorithms and how the resource allocation problem can be solved. In Chapter 6 we describe our system model designed for network management including environment structure and reward functions. The results from the experiments and modeling are presented in Chapter 7, where the performance of the methods is investigated. We discuss issues related to our learning methods in Chapter 8 and also provide potential extensions for future work. Finally, we give a brief summary of the entire work of the thesis and make some final remarks.

2 Related Work

Mobile networking and ML tasks are mostly studied separately. Crossings between DL [7] and wireless networking appeared only recently [3–5, 8, 9]. In [4] and [8], the authors present a comprehensive survey on the crossover between DL and mobile and wireless networking, where they provide an overview of DL-based communication and networking applications categorized by different domains.

One of the most critical ML research paths that has a major impact on the Artificial Intelligence (AI) development is RL [1]. The learning agent in RL is partially inspired by the psychology of human learning. The agent focuses on interaction with the environment by trying different actions and reinforcing actions with more rewarding results. Several works apply RL techniques for network management. A RL-based resource management scheme for clouds robotic was proposed in [10]. In this paper the scheduling problem was formulated as Semi-Markov Decision Process (SMDP) with the aim of allocating cloud computing resources for the robots' efficient service in the system. Q-learning algorithm was implemented in [11] for autonomic cloud resource management. In this work the authors solve the convergence problem by providing a good initial estimate for the Q-function, which speedups learning convergence throughout the training phases. A significant amount of research work has been done on network slicing management and orchestration using RL methods. [12] presents a Q-learning-based dynamic resource adjustment algorithm that aims at maximizing the profit of tenants while ensuring the QoS requirements of end-users.

Nonetheless, RL faces some difficulties in dealing with large state space as it is hard to go through each state and a value function for each state-action pair. Therefor combination of RL with DL [7], referred to as DRL [6], has attracted much attention in recent years. The DRL techniques are capable of providing a good approximation of the objective value while dealing with very large state and action spaces. A variety of projects applying DRL, such as the Go game [13] and Atari video games [14], have achieved an amazing performance.

The DRL is widely implemented in different communication scenarios, such as cellular networks [15], network slicing [16–18], edge computing [19] and vehicle to vehicle (V2V) communications [20]. The Deep Q Learning (DQL) has been used in [18] to solve a standard resource management network slicing scenario, which involves radio resource slicing and core network slicing. In [21] a distributed dynamic power allocation scheme is developed on the basis of model-free DRL technique. In [16] the authors propose a two-level framework for radio resource virtualization and allocation in 5G. In this paper, the RL algorithm autonomously adjusts the assigned resource to slice based on feedback from the users' average QoS utility and average resource usage. In [19] the authors design an intelligent DQL agent at the edge computing node to develop a real-time adaptive policy for computational resource allocation for offloaded tasks of multiple users.

Inspired by the achievements of DRL in solving control problems, there has been increased interest in applying advanced DRL frameworks for problems in wireless communication. Different multi-agent RL scenarios are studied in [21, 22]. In [21] the authors are focused on a multi-agent scenario where each transmitter collects channel state information (CSI) and QoS information from several neighbors and adjusts its own transmit power accordingly. The objective is to maximize a weighted sum-rate utility function [21].

Following the initial success of the DQN [23], a number of improved models have been published successively. The D-DQN was suggested in [24] to eliminate overestimation by adding a target Q network. The Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) have been shown to be efficient in the processing of sequence data with partially observable states [25]. In [26] the authors replaced the last layer in the DQN with an LSTM layer, refereed as Deep Recurrent Q-Network (DRQN). They evaluated the performance of DRQN on standard Atari games.

Other typical DRL framework is the AC method [27]. The AC method is studied in [28–30]. For instance, in [30] the authors propose AC based framework for dynamic multichannel access. They consider both single and multi-user scenarios, where the proposed framework is employed as a single agent in the single-user case, and is extended to a decentralized multi-agent framework in the multi-user case. The AC-based algorithm was suggested in [28] in order to improve the performance of resource allocation in optical networks. The paper implements the AC algorithm in order to find the optimal policy for user scheduling and resource allocation in HetNets [31].

3 5G Network Slicing

This chapter is a short introduction to 5G and Network Slicing. We provide a comprehensive summary of 5G services' classification and requirements.

The rapidly increasing number of mobile devices, causing the explosive growth in capacity, coverage and higher data rates demands, are pushing the evolution of the current mobile communication systems to new generation, i.e. 5G. 5G is bringing many benefits to both customers and service providers. The 5G technology is expected to meet various user QoS requirements in different application scenarios, in terms of data transmission rate and latency. 5G-linked technologies are revolutionary in terms of their effect on the communications network: extreme mobile broadband, wide range of frequency bands, ultra-low latency, wide-area coverage and sliceability. A network slice is defined as a logical (virtual) network, which is built on top of a common infrastructure layer and based on a network resource sharing. Network slicing aims to split the entire network into different parts, each consisting of an end-to-end composition of network resources tailed to meet the customized performance and economic service or customer application requirements.

However, resource management on network slices involves more challenging technical issues, since for Radio Access Network (RAN) it is critical to guarantee the Spectrum Efficiency (SE), while for core network, virtualized functionalities are limited by computing resources. 5G networks should be able to provide dense wide-area coverage with high capacity under the constraints of low power consumption and low cost per device. Extremely low latency and high reliability of 5G networks are required to support a variety of use cases and application scenarios. Therefore, nowadays network slicing is an emerging business to operators and allows them to sell the customized network slices to various tenants at different prices.

According to a global commitment, 5G wireless systems will support three generic services with extremely heterogeneous requirements: Extreme Mobile Broadband (eMBB), Massive Machine Type Communication (mMTC) and Ultra-reliable and low-latency communications (URLLC) [32]. The specifications of the three services are listed in more detail below.

The eMBB service requires both high data rate secure connections with low latency in some area, as well as reliable broadband access over large areas. It is characterized by large payloads and by device activation pattern that remains stable over an extended period of time. The eMBB service aims to maximize the data rate while maintaining a steady reliability with a packet error rate (PER) in the order of 10^{-3} [32].

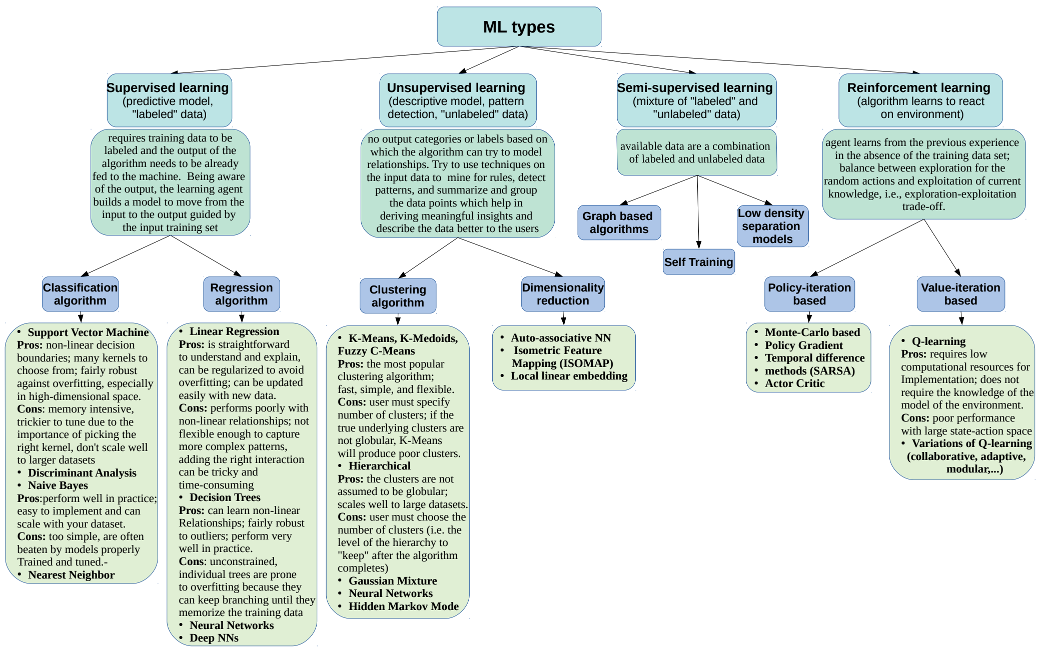
mMTC supports a large number of Internet of Things (IoT) devices, that are only active intermittently and send small payloads of data. mMTC services require wireless connectivity for massive device deployment. mMTC devices typically use low transmission rates in the uplink. At a given time only an unknown(random) number of mMTC devices connected to a given base station (BS) may become active and attempt to send their data. In this case individual resource allocation is infeasible, instead, it is necessary to provide resources that can be shared through random access. Usually the target PER of a single mMTC transmission is low, e.g., in the order of 10^{-1} [32] .

URLLC covers all services requiring ultra-low latency transmissions of small payloads with certain level of reliability from a limited set of terminals. URLLC transmissions are also intermittent, but the number of potentially active URLLC devices is much smaller than for mMTC. Supporting intermittent URLLC transmissions requires a combination of scheduling, so as to ensure a certain amount of predictability in the available resources and thus support high reliability; as well as random access, in order to avoid that too many resources being idle due to the intermittent traffic. Due to the low latency requirements, a URLLC transmission should be localized in time. Diversity, which is critical to achieve high reliability, can hence be achieved only using multiple frequency or spatial resources. The URLLC transmission rate is relatively low, and the main requirement is to guarantee a high level of reliability, with the PER typically below 10^{-5} [32].

Based on the fact that the same physical infrastructure, agile and programmable network architecture is not capable of supporting all three types of services; each service should have a tailor-made network instance to fulfill its requirements.

4 Reinforcement Learning

This section is a comprehensive summary of some basic RL theory. We first present an overview of ML types. We continue with fundamental knowledge of RL and Markov Decision Process (MDP), which are important branches of ML theory. Afterwords we discuss the Q learning technique, that is implemented in our work.


ML types
Supervised learning (predictive model, "labeled" data)
Unsupervised learning (descriptive model, pattern detection, "unlabeled" data)
Semi-supervised learning (mixture of "labeled" and "unlabeled" data)
Reinforcement learning (algorithm learns to react on environment)
Classification algorithm
Regression algorithm
Clustering algorithm
Dimensionality reduction
Graph based algorithms
Self Training
Low density separation models
Policy-iteration based
Value-iteration based

requires training data to be labeled and the output of the algorithm needs to be already fed to the machine. Being aware of the output, the learning agent builds a model to move from the input to the output guided by the input training set

no output categories or labels based on which the algorithm can try to model relationships. Try to use techniques on the input data to mine for rules, detect patterns, and summarize and group the data points which help in deriving meaningful insights and describe the data better to the users

available data are a combination of labeled and unlabeled data

agent learns from the previous experience in the absence of the training data set; balance between exploration for the random actions and exploitation of current knowledge, i.e., exploration-exploitation trade-off.

- **Support Vector Machine**
Pros: non-linear decision boundaries; many kernels to choose from; fairly robust against overfitting, especially in high-dimensional space.
Cons: memory intensive, trickier to tune due to the importance of picking the right kernel, don't scale well to larger datasets
- **Discriminant Analysis**
- **Naive Bayes**
Pros:perform well in practice; easy to implement and can scale with your dataset.
Cons: too simple, are often beaten by models properly Trained and tuned.-
- **Nearest Neighbor**

- **Linear Regression**
Pros: is straightforward to understand and explain, can be regularized to avoid overfitting; can be updated easily with new data.
Cons: performs poorly with non-linear relationships; not flexible enough to capture more complex patterns, adding the right interaction can be tricky and time-consuming
- **Decision Trees**
Pros: can learn non-linear Relationships; fairly robust to outliers; perform very well in practice.
Cons: unconstrained, individual trees are prone to overfitting because they can keep branching until they memorize the training data
- **Neural Networks**
- **Deep NNs**

- **K-Means, K-Medoids, Fuzzy C-Means**
Pros: the most popular clustering algorithm; fast, simple, and flexible.
Cons: user must specify number of clusters; if the true underlying clusters are not globular, K-Means will produce poor clusters.
- **Hierarchical**
Pros: the clusters are not assumed to be globular; scales well to large datasets.
Cons: user must choose the number of clusters (i.e. the level of the hierarchy to "keep" after the algorithm completes)
- **Gaussian Mixture**
- **Neural Networks**
- **Hidden Markov Mode**

- **Auto-associative NN**
- **Isometric Feature Mapping (ISOMAP)**
- **Local linear embedding**

- **Monte-Carlo based**
- **Policy Gradient**
- **Temporal difference**
- **methods (SARSA)**
- **Actor Critic**

- **Q-learning**
Pros: requires low computational resources for Implementation; does not require the knowledge of the model of the environment.
Cons: poor performance with large state-action space
- **Variations of Q-learning (collaborative, adaptive, modular,...)**

Figure 4.1 – ML types.

Recently, ML techniques have been used in a wide variety of applications, such as computer vision, networking and data mining. ML algorithms are able to learn to make decisions directly from data without using explicit instructions. As it's shown on Figure 4.1, existing ML algorithms can be grouped into four categories: supervised learning (SL), unsupervised learning (USL), semi-supervised learning and RL.

4.1 Introduction to Reinforcement Learning

In RL an agent learns to interact with an unknown environment with the goal to maximize the final cumulative reward. The reward is the feedback from the environment resulting from the action made at each step. The agent is not told which actions to take, instead it learns to act and optimize its behavior from the reward obtained from its actions. The learner must figure out which actions produce the most reward by trying them.

Figure 4.2 shows a general architecture of RL that illustrates agent-environment interaction. The agent takes an action a_t based on the current state s_t and policy π, resulting the shift to the next state of the environment s_{t+1}. Subsequently, the environment responses to the taken action in the form of a reward. The reward can either be positive or negative. An agent continuously receives a new state s_{t+1} and reward r_{t+1} every time step t and gradually learns to evaluate the policy and to take the best action.

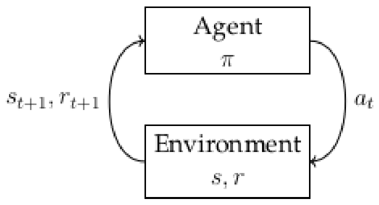


Figure 4.2 – The basic RL scheme

Unlike to SL, RL algorithms are self-learning, they do not need labelled output to learn the model. The goal of RL is to train a smart agent so that it can determine the best actions from its own experience. One of the main challenges of learning is to trade-off between the exploration and the exploitation. In order to achieve a high long term reward, the agent should alternate between exploiting the current knowledge of the environment to maximize the rewards (take greedy actions), and gain new knowledge to discover better action selections (exploring actions).

4.1.1 Elements of Reinforcement Learning

There are four main elements of RL system: a policy, a reward function, a value function and a model of the environment.

The **policy** is stochastic mapping from perceived states of the environment to actions to be taken when in those states. The policy is the core of a RL agent in the sense that the learning behavior is dictated by the policy selected by the agent. In some cases, the policy may be a simple function or a lookup table, and in others it may require comprehensive computing.

The **reward function** defines the goal in a reinforcement learning problem. It is a mapping of each state–action pair of the environment to a single numerical reward. The function should differentiate between good and bad events for the agent. It indicates the immediate reward for actions. It is very convenient to change the optimization goal just by changing the reward function in the learning model, which is impossible with a traditional heuristic algorithm. In general, reward functions may be stochastic.

The **value function** represents the value of a state for an agent to be in. While a reward function shows what is good in the immediate sense, a value function determines long-term payoff of a state. It is equal to expected accumulated future reward for an agent starting from state s. A state with a low immediate reward may still have a high value because other states that offer high rewards may follow it. The value function depends on the policy by which the agent chooses actions to be performed.

There are two types of RL techniques: **model-free** and **model-based**. The **model** represents the behavior of the environment. For example, the model could predict the resulting next state and next reward given a current state and action. Models are used for planning, where any way of choosing a course of actions is made by considering possible future situations. The model of the system is represented with transition probabilities between all states and the reward function for each of the states. Model-free learning techniques, such as Q-learning, learn the state value function and obtain an optimal policy without the need of any model (directly from experience).

4.1.2 ϵ-greedy policy

Exploration vs exploitation dilemma is crucial in RL. To converge to the optimum policy, the learning algorithm must try out all possible alternatives. This includes both exploration and exploitation. Exploitation means taking action that appears best according to the knowledge already learned, while exploration means trying new things out in the hope that better actions will be found. To maximize the reword, the agent has to exploit and choose the best action he knows. However,the exploitation doesn't guarantee the optimal solution in the long run because the actions chosen may not be the best ones, and there can be other unexplored actions that result in better long-term reward. The agent must do a trade-off between acting optimally in the context of current knowledge and acting to obtain more knowledge. The ϵ-greedy approach provides a solution to this problem by using ϵ to determine the randomness in action selections and control the amount of exploration. Usually ϵ is is in range of 0 and 1. The agent chooses the best action with probability ϵ, and otherwise, it chooses the actions randomly:

$$a_t = \begin{cases} \max_a Q(s_t, a) & \text{with probability } \epsilon \\ \text{random action} & \text{otherwise} \end{cases} \tag{4.1}$$

At the beginning of the training, the agent does only exploration ($\epsilon = 1$), and as the training advances, we reduce ϵ so the agent exploits learned information and uses more its policy.

4.2 Reinforcement Learning types

There are three types of RL algorithms: policy-based, value-based, and actor-critic methods. Policy-based methods, such as REINFORCE family, directly parameterize the policy $\pi(a|s;\theta)$ with parameter θ, which is updated to improve $\mathbb{E}[R_t]$ for maximum return. Policy-based approaches can find stochastic optimal policies efficiently and have better convergence properties. Typically, however, they converge to the local optimum, and it is ineffective and has large variance to evaluate a policy. Value-based methods, such as Q-learning, SARSA, generally use temporaral difference iteration to estimate a policy's expected rewards and select action with the highest value function. Value-based approaches usually quantize the continuous action space and add quantization noise, which can lead to oscillations or non-convergence when conducting this process iteratively, and thus they are unable to find the true optimal policy for problems with the continuous-valued action space.

Therefore, actor-critic RL algorithms are proposed to combine the process of policy-based and value-based methods. We will discuss actor-critic methods later.

4.3 Markov Decision Process

In a typical RL problem the learner and decision-maker is called the agent. The surrounding it interacts with, including everything outside the agent, is called the environment. In return, the environment provides rewards and a new state based on the agent's actions. After considering the design of the problem, we can move into the mathematical framework to solve the RL problem. This is where the MDP [1] comes in. MDP is a mathematical model of a system defined by its state and action sets and by the one-step dynamics of the environment. MDP can be formulated by a 5-tuple as $M = [S, A, p(s'|s,a), r, \gamma]$, where S denotes a finite state space, A stands for an action set, $p(s'|s,a)$ indicates the probability of each possible next state s' given any state and action, s and a, r is an immediate and $\gamma \in [0,1]$ is a discount factor reflecting the decreasing value of current reward on future ones.
Given any current state s, action a and next state s', the expected value of the next reward is:

$$r(s,a,s') = \mathbb{E}[R_{t+1}|S_t = s, A_t = a, S_{t+1} = s'] \tag{4.2}$$

Similarly, the transition probability can be defined as:

$$p(s'|s,a) = \mathbf{Pr}[S_{t+1} = s'|S_t = s, A_t = a] \tag{4.3}$$

In RL the agent's goal is to find a policy $\pi(s,a) \in \Pi$, so as to optimize the state-value function $V^{\pi}(s) : S \to R$ such that:

$$V^{\pi}(s) = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k r_{t+k}|s_t = s, \pi] \tag{4.4}$$

In addition to the V-value function, there is another valuable measure, which indicates how good the current state and an action performed in that state given that policy. This is known as the action-value function, $Q^{\pi}(s,a) : S \times A \to \mathbb{R}$ given by:

$$Q^{\pi}(s,a) = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, a_t = a, \pi] \tag{4.5}$$

By combining Q^{π} and V^{π} we can calculate how good the action a is, as compared to the expected return when following the policy π. It is also possible to define the advantage function:

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s) \tag{4.6}$$

MDP provides a good starting point for addressing and investigating the RL problem. Even though the actual environment do not comply with all requirements, it is often a good approximation. The developer attempts in practice to adjust the problem to suit the MDP framework.

4.4 Q-learning

Q-learning is a greatly used model-free RL algorithm where action-value function $Q(s,a)$ is iteratively updated and is preserved in a lookup table one entry for every state-action pair. The action-value function estimates the expected utility of taking an action in a given state. The Q-learning algorithm uses the Bellman equation to learn the optimal Q-value function:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[s_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{4.7}$$

where r_j is the reward received when moving from state s_j to the state s_{j+1}, and α is a learning rate $(0 < \alpha \leq 1)$, γ is a discount factor. γ is a value between 0 and 1, and has the effect of valuing rewards received earlier higher than those received later. The Q learning architecture is presented on Figure 4.3.

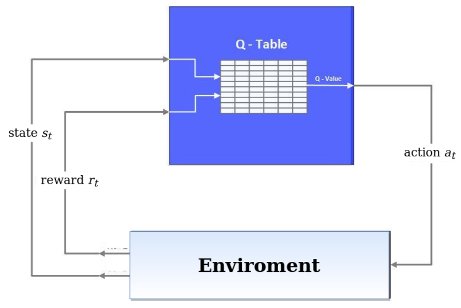


Figure 4.3 – Q-learning architecture

Algorithm 1 Q-learning based on [1]

```
1: Initialize $Q(s, a)$ arbitrarily
2: for episode = 1 to $M$ do
3:     Initialize $s$
4:     for step $t$ = 1 to $T$ do
5:         Choose $a_t$ from $s_t$ using policy derived from $Q$
6:         Take action $a_t$ , observe $r_t$ , $s_{t+1}$
7:         $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[s_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$
8:         $s_t \leftarrow s_{t+1}$
9:     end for
10: end for
```

This approach is often not feasible in real applications due to the high-dimensional (possibly continuous) state-action space. In this context, use of function approximations is a common approach to the limitations of tabular methods. One example is to represent the Q-function with a function approximation $Q(s, a, \theta)$, as in the case of DQL, where θ refers to a weight vector defining the parameterized function.

5 Deep Reinforcement Learning

Many networking problems can be formulated as MDP, where RL can play a key role. Nevertheless, some of these problems include high-dimensional inputs, reducing the applicability of conventional RL algorithms. The DRL-based techniques can solve this issue by training deep NNs-based Therefore, the application of DRL promises to address network management and control issues in complex mobile environments. This chapter starts with a section about NNs and their basic properties. It will continue with a description of DRL algorithms, including the algorithms applied in this thesis.

5.1 Neural Networks

The structure of the NN is similar to the process of perception in a brain where, given the current context, a specific set of units is triggered, affecting the performance of the neural network model. The key objective of Deep Neureal Network (DNN)s is to approximate complex functions by composing simple and pre-defined unit (or neuron) operations. Depending on the model structure, the operations performed are usually defined by a weighted combination of a specific group of hidden neurons organized in layers with a non-linear activation function. The most basic NN is the Feed-forward NN. This implies that the neurons are arranged into layers, where every neuron of one layer is connected to every unit of the next layer only in one direction, from the input layer to the output layer as shown on Figure 5.1. Every neuron receiving multiple inputs takes a weighted sum of them, passes it through an activation function, and responds with an output. Usually, there are no connections in a single layer between neurons.

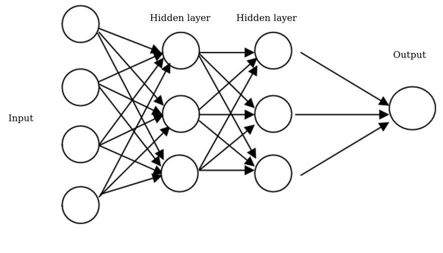


Figure 5.1 – Feed-forward NN architecture

Deep feed-forward NNs with multiple hidden layers between input and output layer are capable of performing a generalization task and reducing the complexity of the state space.

5.2 Deep Q-Network

Calculating the Q-value of each state-action pair, as discussed above, becomes computationally infeasible as the complexity of the environment (and thus the spaces of state and action) increases. To address this issue, we can use a framework of machine learning to serve as a Q-value function approximator. A neural network parameterizes the Q-values, the weights and biases of which are implied in the new Q-value:

$$Q^*(s,a) \approx Q(s,a;\theta) \tag{5.1}$$

Neural networks have been shown to be effective for function approximation in learning tasks.

Nevertheless, DQN's performance does not rely solely on the use of a neural network approximator function. There are some learning stability problems, that prevent neural networks as nonlinear function approximators from converging. The DQN method introduces a target Q-network to stabilize learning and reduce the variance of the approximated function. Target Q-network only copies the parameters from the original trained Q-network after hundreds or thousand training steps and therefore does not change quickly and allows the algorithm to learn stable long term dependencies (Mnih, Kavukcuoglu, Silver, Rusu, et al. 2015). With parameters $\widehat{\theta}$ of the target network, the square loss function can be represented as:

$$L_{DQN}(s_t, a_t, r_{t+1}, \theta, \widehat{\theta}) = (r_{t+1} + \gamma \max_a Q(s_{t+1}, a; \widehat{\theta}) - Q(s_t, a_t; \theta))^2 \tag{5.2}$$

In DQL the state can be provided as an input to the Q-network and a different output is given for each of the possible actions. This provides an efficient structure that has the advantage of obtaining the computation of $\max_{a'} Q(s', a'; \theta_i)$ in a single forward pass in the neural network for a given state. Unfortunately, due to the inherent structure of the learned data, simple gradient descent on the loss function with the target network can still result in high variance of the function estimator. The use of experience replay is one of the key concepts brought by Mnih work [23]. This method stores the agent's experience (transitions between states in the past, and the corresponding actions and rewards) at each recorded time step $e_t = [s_t, a_t, r_t, s_{t+1}]$ in a replay memory in order to be able to calculate the loss correctly in the future at any time step. Mini-batches of experiences are then sampled at random and used to perform the weight updates for the DQN training. Q-learning with experience replay offered many advantages over Q-learning's usual form. This makes it possible for the network to use every stored experience for many updates, which makes the learning process more efficient. Experience replay reduced the variance of the updates by eliminating the correlation between these samples. It has been shown that this mechanism improves the stability of the training.

Algorithm 2 DQN and D-DQN algorithms based on [1]

```
1: Initialize replay memory $D$
2: Initialize action-value function $Q$ with random weights $\theta$
3: Initialize target action-value function $\widehat{Q}$ with random weights $\widehat{\theta} = \theta$
4: for episode = 1 to $M$ do do
5:     Initialize $s_1$
6:     for step $t$ = 1 to $T$ do
7:         $a_t = \begin{cases} \max_a Q(s_t, a; \theta) & \texttt{with probability } \epsilon \\ \texttt{random action} & \texttt{otherwise} \end{cases}$
8:
9:         Take action $a_t$, observe $r_t$, $s_{t+1}$
10:        Set $s_t = s_{t+1}$
11:        Store transition $(s_t, a_t, r_t, s_{t+1})$ in $D$
12:        // sample from experience replay memory
13:        Sample random minibatch of transitions $(s_t, a_t, r_t, s_{t+1})$ from $D$
14:        Perform a gradient descent step on $(r_{t+1} + \gamma \max_a Q(s_{t+1}, a; \widehat{\theta}) - Q(s_t, a_t; \theta))^2$
    w.r.t. to the network parameters $\theta$
15:        // update target network
16:        Every $K$ steps reset $\widehat{Q} = Q$, set $\widehat{\theta} = \theta$
17:    end for
18: end for
```

5.2.1 Double DQN

Many new improvements have been made since DQN was introduced, which show better learning properties. The D-DQN is one of the most significant enhancements.

D-DQN as proposed in [24] is able to improve the performance of DQN applied to Atari games by a minor modification of the training target. Q-learning max operation uses the same values for selecting and evaluating an action. So in case of inaccuracies or noise, Q-learning algorithm tends to overestimate Q-values, resulting in overoptimistic action value estimates. The use of two networks in DQN leads to less overestimation of the Q-learning values. The loss function with target network given in equation 5.2 is considered problematic, because it still tends to overestimate the future action values. Both the selection and evaluation of future actions depend on the parameters of the target network $\widehat{\theta}$. The loss function used by D-DQN disentangles the action selection from the evaluation of the selected action by using the trained parameters θ to select future actions instead of those of the target network:

$$L_{D-DQN}(s_t, a_t, r_{t+1}, \theta, \widehat{\theta}) = (r_{t+1} + \gamma Q(s_{t+1,a} Q(s_{t+1}, a; \widehat{\theta}); \widehat{\theta}) - Q(s_t, a_t; \theta))^2 \tag{5.3}$$

5.3 Actor-Critic

All of the approaches we have considered so far have learned the values of state–action pairs, and then used those values explicitly to implement the policy and choose actions.

All methods of this form can be called action-value methods. Now we explore methods that are not action-value methods.

Techniques that learn approximations to both policy and value functions are often referred to as actor-critical techniques, where ' actor ' is a reference to the policy learned, and ' critic ' refers to the learned value function. Actor – critic methods are methods with a separate memory structure to represent the parametrized policy independently from the value function. The actor is the function estimator, which is used to select actions. The critic is an estimated value function, it criticizes the actions made by the actor. Learning is always on-policy: the critic needs to learn and criticize whatever policy the actor is currently pursuing. Both parts can be modeled by neural networks.

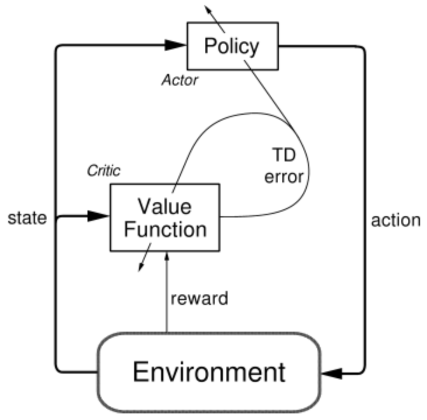


Figure 5.2 – Actor-critic architecture [1]

Figure 5.2 shows the interactions of the two function estimators. The critique takes the form of a TD error. This scalar signal is the sole output of the critic and drives all learning in both actor and critic. Typically, the critic is a state-value function. After each action selection, the agent will execute it in the environment and send the current observation along with the feedback from the environment to the critic. The feedback includes the reward and the next time instant observation. Then, the critic calculates the Temporal Difference (TD) error:

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \tag{5.4}$$

where V is the current value function implemented by the critic. The critic is updated by minimizing the least squares temporal difference. After the critic updates the value function approximation and its parameters, the actor uses the the output of critique to update it's policy. TD error evaluates the learning process and drives all learning in both actor and critic. It is used to adjust both the actor and the critics in the direction that would mostly minimize the error. Therefore, actor-critic methods typically have good convergence properties [1].

6 System Model

In this section, we describe our system model and the basics of the proposed resource allocation framework.

6.1 Model Parameters

The Table 6.1 summarizes the notation adopted in this work. Let I be a set of services running over the resources of a 5G Network Service Provider (NSP). Every service is offered over a separate network *slice* , i.e. a virtual network composed by a subset of virtual and physical resources of the NSP including routers, switches and bandwidth resources in both the core network and the RAN segments controlled by the NSP. Let t_j be the j-th *time frame* for which the resource scheduler of the NSP tries to distribute its resources to its client slices.

I	set of service slices
N	number of slices
$\bar{x_i}(j)$	the actual RB' demand of slice i
$x'_i(j)$	the actual allocation of RB
$\hat{x}_i(j)$	the estimated RB' demand
$x'_{i,min}$	the guaranteed resources for the slice i (reflected by the Service Level Agreement (SLA) of the slice)/ minimum allowed allocation
$\bar{y_i}(j)$	the actual demand of processing power
$y'_i(j)$	the actual allocation of processing power
N_{RB}	total number of available RB
L_i	maximum allowed delay for slice i
$D_i(j)$	computed delay caused by the allocation of resources
T	number of time slots
w	power to RB ratio

Table 6.1 – Parameters used in the model

For further system description we need to define difference values between number of demanded and allocated resources:

$$\Delta x_i(j) = \bar{x_i}(j) - x'_i(j) \geq 0 \tag{6.1}$$

$$\Delta y_i(j) = \bar{y_i}(j) - y'_i(j) \geq 0 \tag{6.2}$$

To simplify the system model we're introducing w as a power to bandwidth (RB) ratio:

$$y'_i(j) = w \times x'_i(j) : \sum_{i=1}^{n} y'_i(j) \leq 1 \tag{6.3}$$

$$\sum_{i=1}^{n} y'_i(j) = \sum_{i=1}^{n} w \times x'_i(j) \leq w \times B \tag{6.4}$$

For time slot j the estimated demand is computed from previous demands using:

$$\hat{x}_i(j) = \theta \bar{x}_i(j-1) + (1-\theta)\frac{1}{j-2}\sum_{n=1}^{j-2} \bar{x}_i(n) \tag{6.5}$$

where θ is a number between 0 and 1, and has the effect of valuing resources' demand received earlier higher than those received later. We used $\theta = 0.9$.

6.2 Monetary gain function

Let the total monetary gain T_i for a slice instance i be

$$T_i(x'_i, \bar{x}_i, y'_i) = M_i(x'_i) - \Phi_i(x'_i, \bar{x}_i) - \Gamma_i(x'_i, \bar{x}_i, y'_i) \tag{6.6}$$

where $M_i(x'_i) \geq 0$ is the *monetary RB fee*; $\Phi_i(x'_i, \bar{x}_i) \geq 0$ is the *RB-related penalty* that it is positive in case the $x'_i < \bar{x}_i$, whereas $\Phi_i(x'_i, \bar{x}_i) = 0 \iff x'_i \geq \bar{x}_i$ (recall that $x'_i \leq \bar{x}_i$ based on (6.1)); and $\Gamma_i(x'_i, \bar{x}_i, y'_i) \geq 0$ is the *delay-related penalty* for the delay caused due to allocated x'_i relative to $\bar{x}_i$, and the allocated processing power y'_i. Thus, the optimization goal can be defined as the maximization of the *total monetary gain*, i.e:

$$max \sum_{i=1}^{n} T_i(x'_i, \bar{x}_i, y'_i) \tag{6.7}$$

with RB' and power constraints:

$$\sum_{i=1}^{n} x'_i(j) \leq N_{RB} \tag{6.8}$$

$$\sum_{i=1}^{n} y'_i(j) \leq 1 \tag{6.9}$$

Based on (6.6) and for a slice instance i, we consider the following:

— The *monetary gain fee* is a *linear function* of the allocated RB x'_i, using a linear factor μ_i: $M_i(x'_i) \equiv \mu_i \times x'_i$.

— The *RB penalty* is quadratic: $\Phi_i(x'_i, \bar{x}_i) \equiv \phi_i \times \Delta x_i^2 = \phi_i \times (\bar{x}_i - x'_i)^2$, where $\phi \geq 0$ is a linear penalty factor.

— The *delay penalty* is also quadratic and defined as:

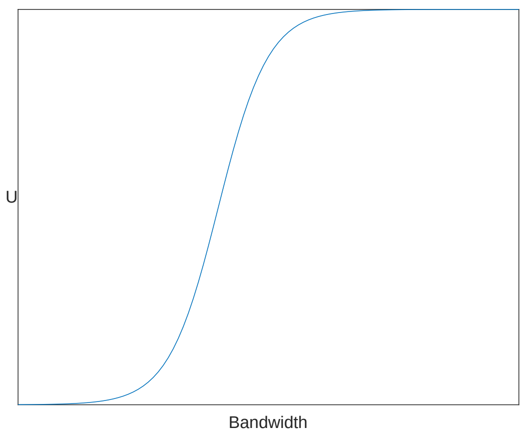
$$\Gamma_i(x'_i, y'_i, j) \equiv \begin{cases} 0, & \text{if } D_i(j) \leq L_i \\ \gamma_i \times (D_i(j) - L_i)^2 \geq 0 & \text{otherwise,} \end{cases} \tag{6.10}$$

where $\gamma_i \geq$ is linear factor; L_i is the *maximum allowed delay* ,above which there would be service degradation, for slice i; and $D_i(j) \equiv D_i(y'_i(j), x'_i(j))$ is the delay caused by the allocation of processing $y'_i(j)$ and RB $x'_i(j)$ resources, which we define as

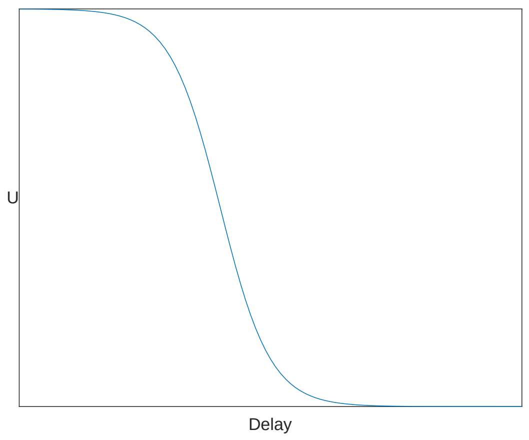
$$D_i(y'_i(j), x'_i(j)) = exp(-y'_i(j)) + g(x'_i(j)), \tag{6.11}$$

Notice that both penalties, namely, $\Phi_i(x'_i, \bar{x}_i)$ and $\Gamma_i(x'_i, \bar{x}_i, y'_i)$, are convex functions (due being quadratic functions), having a global minimum that is equal to zero.

Utility function-based monetary gain and penalties



(a) RB utility



(b) Delay Utility

Figure 6.1 – Both utilities follow a sigmoid utility function. In the case of (a) a maximum value is achieved above a maximum RB threshold, whereas in the case of (b) a maximum value is achieved below a minimum delay threshold.

The values of a utility function expresses the satisfaction level of the slice service for a given RB input to the service. Likewise, there can be a different utility function for the related delay. In this context, let us assume RB and delay utility functions U_x and U_d, as shown in fig. 6.1(a) and fig. 6.1(b), respectively.

We may consider utility functions for the monetary gain and both penalties, assuming that the maximum utility value is $U_x^{max} = U_x(\bar{x_i}) \geq U_x(x_i') \implies U_x(\bar{x_i}) - U_x(x_i') \geq 0$ for the above utilities.

— The *RB monetary gain* is defined as a linear function of the allocated RB utility $M_i(x_i') \equiv \mu_i \times U_x(x')$.

— The *RB penalty* is defined as: $\Phi_i(x_i', \bar{x_i}) \equiv \phi_i \times (U_x(\bar{x}) - U_x(x'))$, where $\phi \geq 0$ is a linear penalty factor.

— Assume that U_d is defined by the inverse sigmoid of fig. 6.1(b).
Assume again (6.11), i.e. that delay d_i is due to the allocated processing power and RB for slice i, as captured by $d_i = D_i(y_i'(j), x_i'(j)) = exp(-y_i'(j)) + g(x_i'(j))$, and that $U_d^{max} = U_d(L_i) \geq U_d(d_i) \implies U_d(L_i) - U_d(d_i) \geq 0$ for some acceptable delay L_i.
Then, the *delay penalty* can be defined as $\Gamma_i(x_i', \bar{x_i}, y_i') \equiv \gamma_i \times (U_d(L_i) - U_d(d_i)) \geq 0$, where $\gamma \geq 0$ is a linear penalty factor.

6.3 Problem formulation

The agent designs an action to perform resource allocation. The goal is to learn a policy, which chooses one action with highest reward for all slices. There are three key elements in the RL, namely state, action, and reward:

State: The state in RL is a space to reflect the situation of the environment. The state consists of N-size vector $(\bar{x}_1(j), \bar{x}_2(j), ..., \bar{x}_N(j))$, which are the demanded amounts of bandwidth/RBs per each slice. The states are defined as all possible combinations of demanded number of RB for N slices. Then number of states for Q-table are calculated as: $N_{states} = N_{RB}^N$.

Action: The objective of an agent is to map the space of states to the space of actions. In this system, the action is a N-size vector $(x_1'(j), x_2'(j), ..., x_N'(j))$, which are the amounts of bandwidth/RBs that the BS allocates to each user. The actions space consists of all possible allocation combinations for N users - combinatorial action space. Then we try to reduce number of actions by putting constraints, such as: sum of allocated RB must be less than number of available RB; $x_{i,min}'$ - minimum allocation implementation. After this number of possible actions is much smaller than number of states. It significantly reduces computational time of program.

Reward: Based on current state and action, the agent obtains a reward from the environment. The reward function of our system is defined in Section 6.2.

6.4 Implementation Details

All implementations purely consist of python code, while numpy is used for general data processing. The agents were tuned and trained on the model using the python framework TensorFlow [33]. We use TensorFlow for neural network training and to build the architecture of the respective networks. For drawing plots and statistics we rely on matplotlib. The choice of optimizer plays an important role in achieving good learning properties due to the highly non-convex problem of optimizing the weights in a NN. Finding the right optimizer and tuning of it can be the difference between getting stuck in a local minima or not, and can also have a major effect on the number of iterations that is required before converging to a good solution.

The Adam step size optimizer introduced in [34] works close to regular stochastic gradient descent/ascent but with an adaptive step size. It ensures the gradient-based weights update adapts to the gradient's characteristics. In particular, the Adam optimizes the step size based on estimates of the gradient's first and second moment. This leads to lower risk of getting stuck in a local optimum when updating the weights. In this thesis, the Adam step optimizer showed good results and was used in the implementation of DQN, D-DQN and AC.

7 Experiments and Results

In this chapter the results of the thesis, based on the implementation described in the previous chapter, are presented, discussed and evaluated. The evaluation focuses on the agents applied on the model with respect to convergence rate, robustness/generalization of policies and the reward.

7.1 Simulation Configuration

We consider a scenario, in which a set of eMBB, mMTC and URLLC devices are connected to a common BS, as shown in Figure 7.1. For all simulations we use synthetic data, which were generated using Poisson, Gaussian and Uniform distributions. Number of available RB is set to 40. Each simulation is 50000 time slots long. In order to analyse the scheme performance for each scenario we computed the final reward values over all time slots.

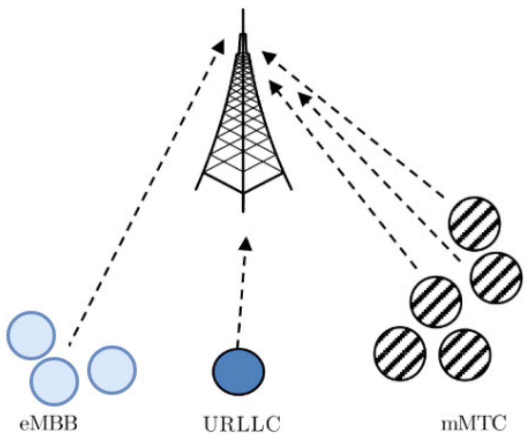


Figure 7.1 – The considered scenario with transmissions to a common BS from devices using the three generic 5G services.

For all four trained agents we run three different simulations. In each case we consider three slices. We first consider a simple scenario (**Test 1**), when the traffic per slice is IID from slot to slot. the amount of demanded resources is slightly higher than the number of available resources. The traffic is simulated with Gaussian (normal) distribution. The mean ("centre") of the distribution is equal to 14 and the standard deviation (spread or "width") of the distribution is 0,5.

In the second scenario (**Test 2**) the traffic per slice is IID, but different between slices. The traffic of first slice is simulated as Gaussian distribution with the mean of 14 and the standard deviation is 0,5. The second slice's demand is presented as the Uniform distribution with samples uniformly distributed over the half-open boundaries of the output interval [10, 12). And the third slice is simulated as Poisson distribution with the mean of 17.

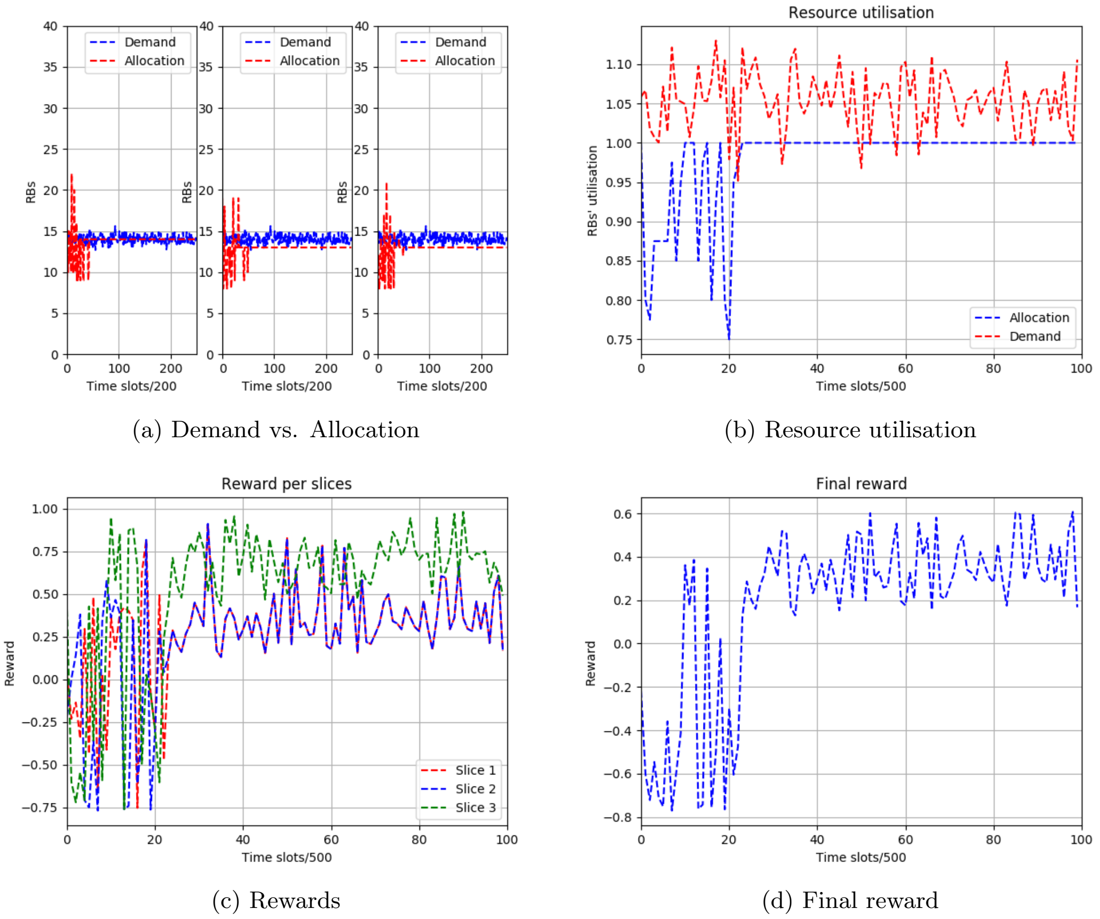
The third scenario (**Test 3**) accounts with the varying traffic patterns. In other words, how traffic demand can change for each slice over time.We address a time-varying environment to identify the adaptive ability of the proposed frameworks. For this case we run the simulation of 100000 time slots long. In this case the traffic demand is simulated randomly. In the end we compare performance of all techniques for every single simulation scenario.

7.2 Tabular Q learning

The Q learning algorithm is discussed in Section 4.4. The number of rows in the Q-table corresponds to the number of states and the number of columns is equal to the number of all possible actions.

Test 1

In this scenario all slices have the same traffic pattern. It's a simple scenario, where we can guess the expected performance. From Figure 7.2(a) we observe that Q learning algorithm tends to allocate the same number of RBs to all slices. The allocation curve at the beginning is highly fluctuating, which is due to training and exploration period. Nevertheless, the algorithm is gradually converging to a fixed allocation.



(a) Demand vs. Allocation

(b) Resource utilisation

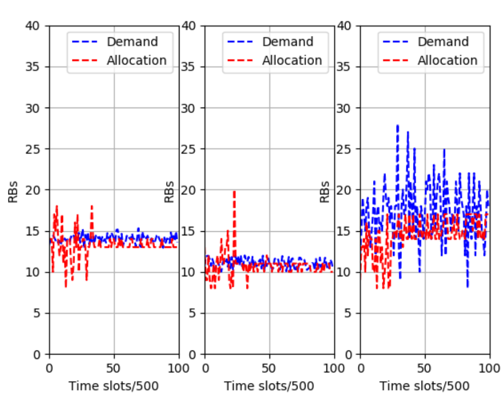
(c) Rewards

(d) Final reward

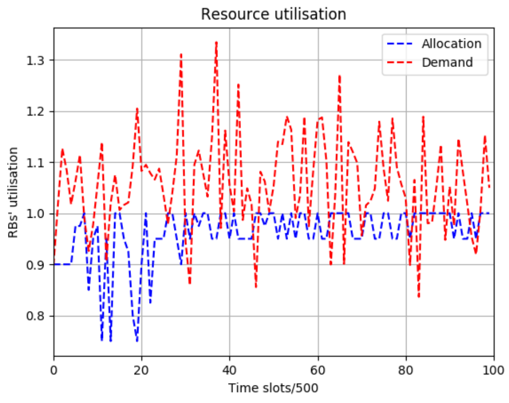
Figure 7.2 – Q-learning performance for resource allocation task in first scenario

Test 2

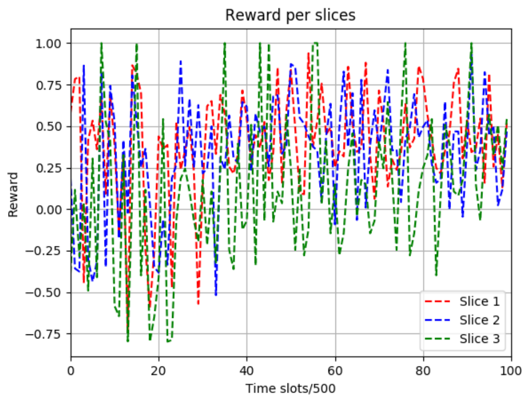
In this scenario the traffic is different between slices. The Q learning algorithm learns a fixed allocation with a small variance over time. Highly varying traffic of slice 3 explains fluctuations in resource utilisation and reward curves.



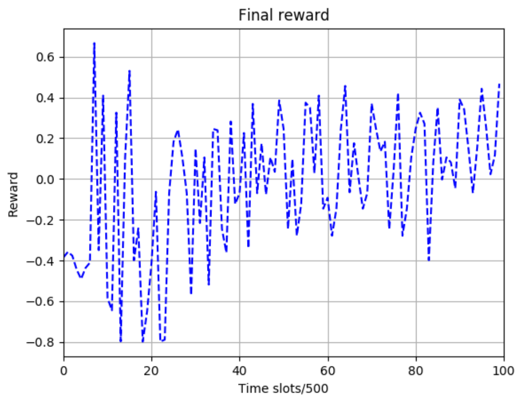
(a) Demand vs. Allocation



(b) Resource utilisation



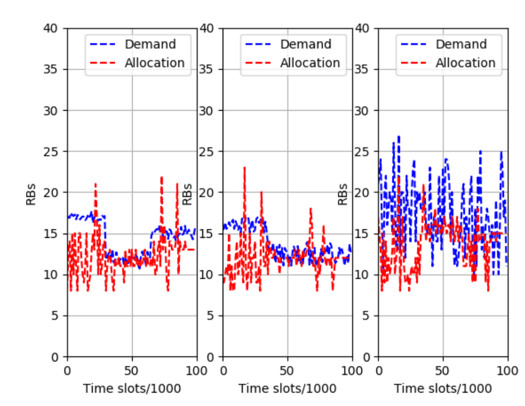
(c) Rewards



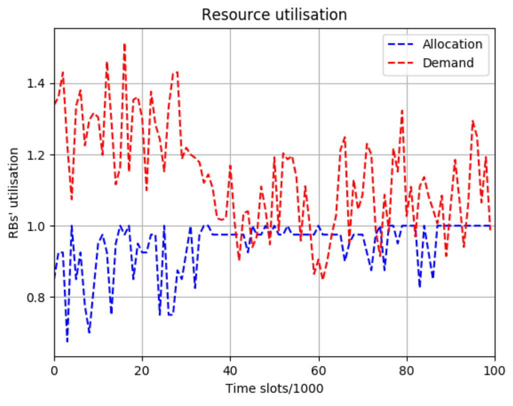
(d) Final reward

Figure 7.3 – Q-learning performance for resource allocation task in second scenario

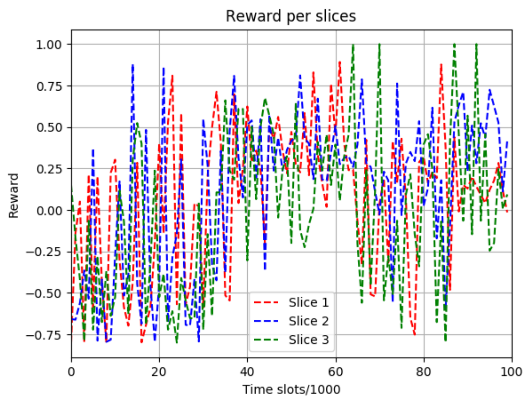
Test 3



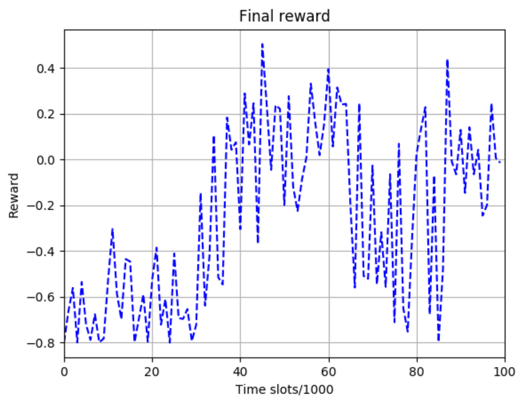
(a) Demand vs. Allocation



(b) Resource utilisation



(c) Rewards



(d) Final reward

Figure 7.4 – Q-learning performance for resource allocation task in third scenario

In Figure 7.4 we can see that Q learning based framework adapts to the time-varying environment, but shows a big variance in both resource utilisation and gained reward over time.

7.3 Deep Q-learning

We design a DQN by following the Deep Q-learning with Experience Replay Algorithm [23] and implement it in TensorFlow [33]. The structure of our DQN is finalized as a fully connected neural network with each of the two hidden layers containing 75 neurons. The activation function of each neuron is Rectified Linear Unit (ReLU), which computes the function $f(x) = max(x, 0)$. The state of the DQN is defined as the combination of previous observations over previous M time slots, which serves as the input to the DQN. A vector of length N is used to represent the observation at a time slot, where each item in the vector indicates the bandwidth demand of the corresponding slice. The output of the DQN is a vector of length equal to the number of actions (possible allocations), where the ith item represents the Q value of a given state if action i is selected. We apply the ϵ-greedy policy with increasing ϵ to balance the exploration and exploitation, i.e., with probability $1 - \epsilon$ the agent selects uniformly a random action, and with probability ϵ the agent chooses the action that maximizes the Q value of a given state. A technique called Experience Replay breaks correlations among data samples and make the training stable and convergent. At each time slot t during training, when an action a_t is taken given the state is s_t , the user gains a corresponding reward r_t and the state is updated to s_{t+1}, a piece of record (s_t, a_t, r_t, s_{t+1}) is stored into a place called replay memory as shown on Figure 7.5. When updating the weights θ of the DQN, a minibatch of 32 samples are randomly selected from the replay memory to compute the loss function, and then a recently proposed Adam algorithm [34] is used to conduct the stochastic gradient descent to update the weights.

Table 7.1 – Parameter setting for DQN and D-DQN

Parameter	**Value**
Q-network	[75, 75] ReLU
Memory size D	2000
Minibatch size M_j	32
Discount factor γ	0.8
Learning rate α	0.01
The period of replacing target Q network k	300
Epsilon start ϵ	0
Epsilon end ϵ_{end}	1
Epsilon increase $\Delta\epsilon$	0.0001

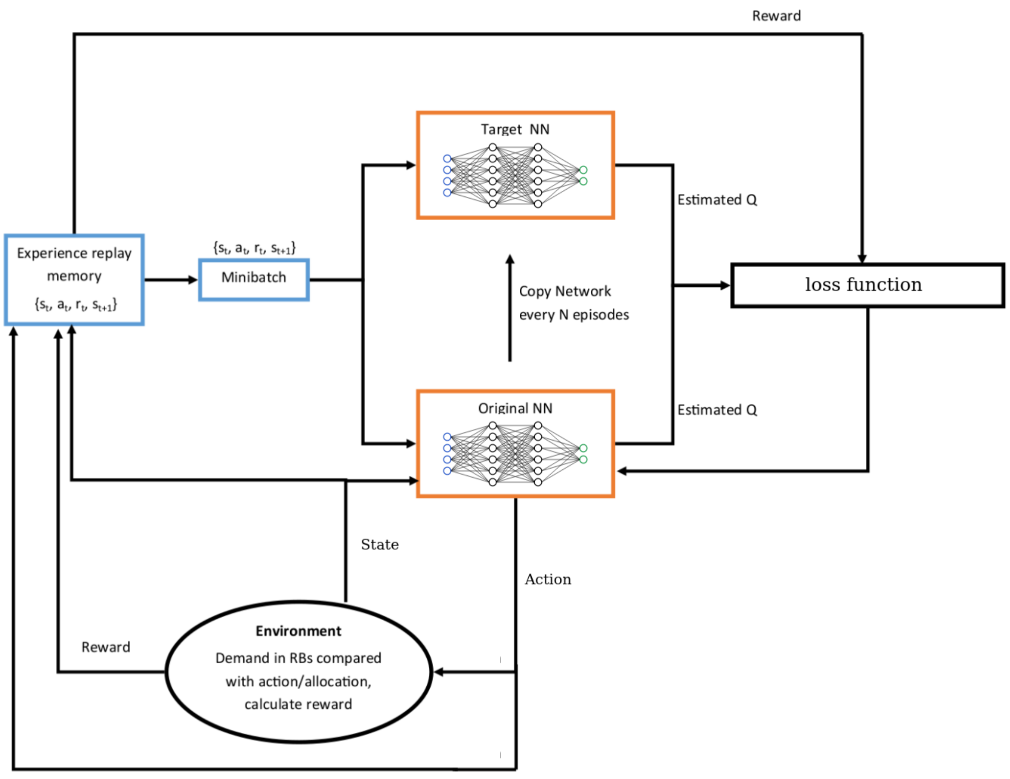
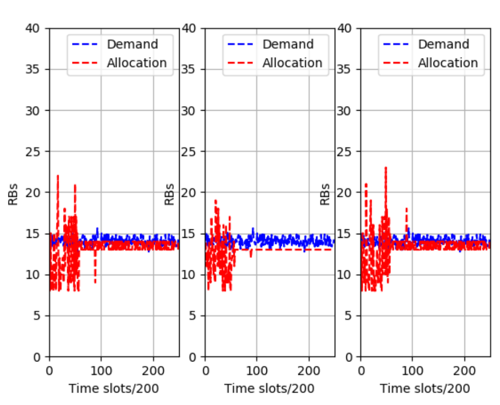
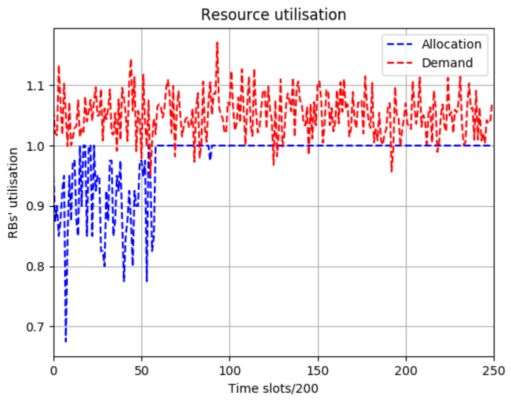

Reward
Target NN
Estimated Q
Experience replay memory
{s_t, a_t, r_t, s_{t+1}}
{s_t, a_t, r_t, s_{t+1}}
Minibatch
Copy Network every N episodes
loss function
Original NN
Estimated Q
State
Action
Reward
Environment
Demand in RBs compared with action/allocation, calculate reward

Figure 7.5 – DQN and D-DQN architecture

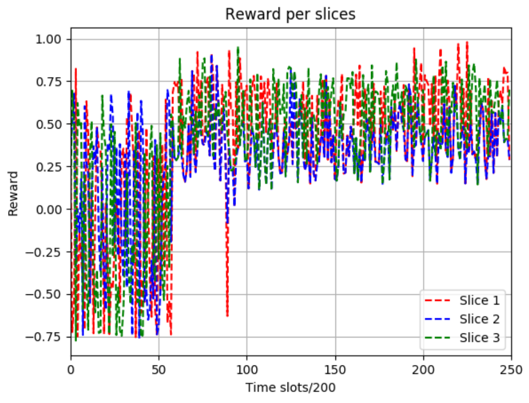
Test 1



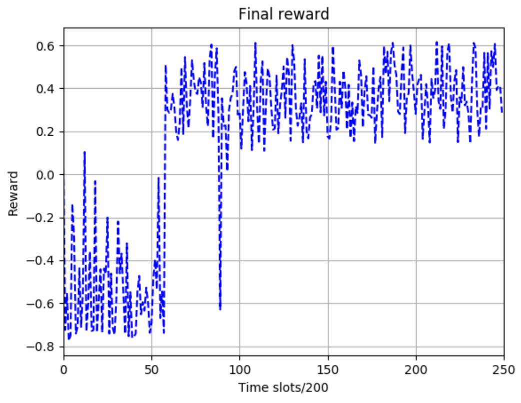
(a) Demand vs. Allocation



(b) Resource utilisation



(c) Rewards



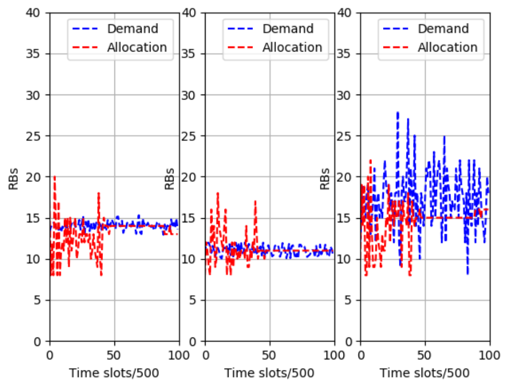
(d) Final reward

Figure 7.6 – DQN performance for resource allocation task in first scenario

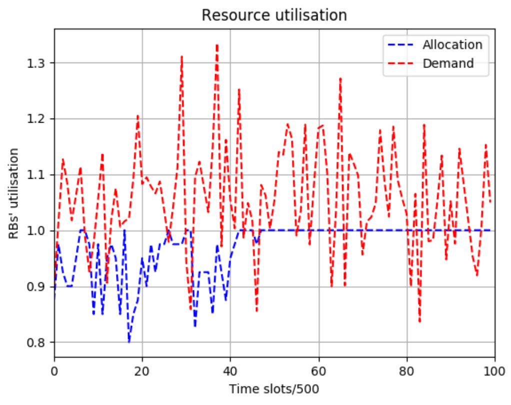
In Figure 7.6 we can see, that the DQN algorithm tries to allocate resources fairly, which is expected. However small fluctuations of allocation curves show that the algorithm is learning noise. It should learn not to have memory or state, i.e. the optimal action should be independent of state.

7.3.1 Test 2

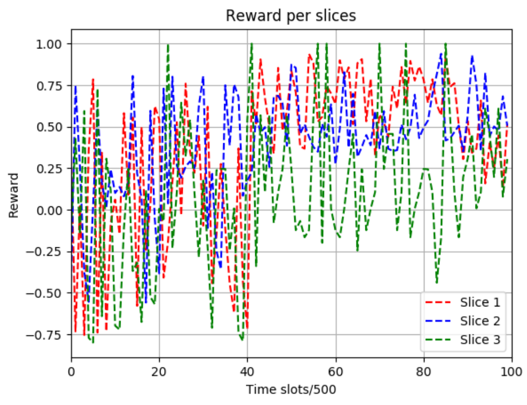
For the case with different IID traffic patterns the DQN algorithm converges to a fixed allocation. The only issue is a long training period, which has a significant impact on the performance of dynamic allocation. On Figure 7.7(c) the reward curve of slice 3 is highly fluctuated comparing with other two slices, which is due to time-varying traffic.



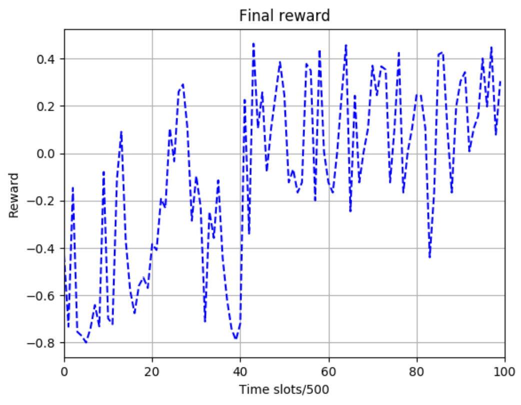
(a) Demand vs. Allocation



(b) Resource utilisation



(c) Rewards

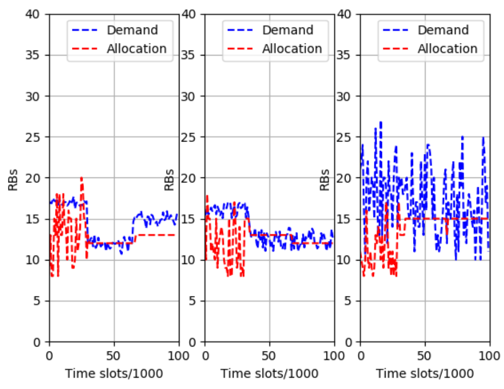


(d) Final reward

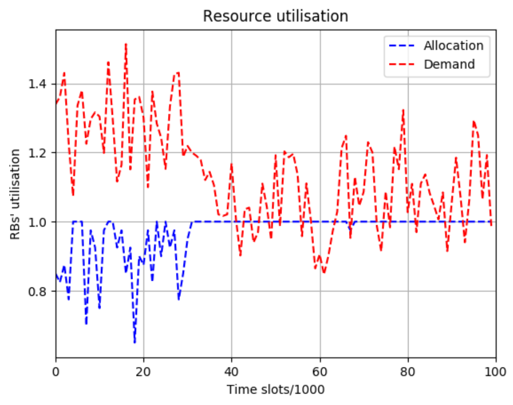
Figure 7.7 – DQN performance for resource allocation task in second scenario

Test 3

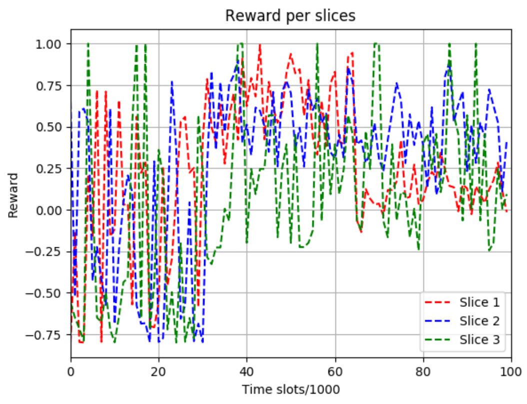
Figure 7.8 shows that DQN learning based framework adapts to the time-varying environment. We can observe a significant performance improvement comparing with Q learning on Figure 7.4.



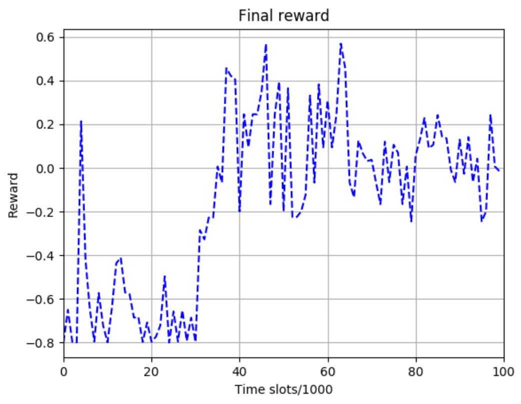
(a) Demand vs. Allocation



(b) Resource utilisation



(c) Rewards



(d) Final reward

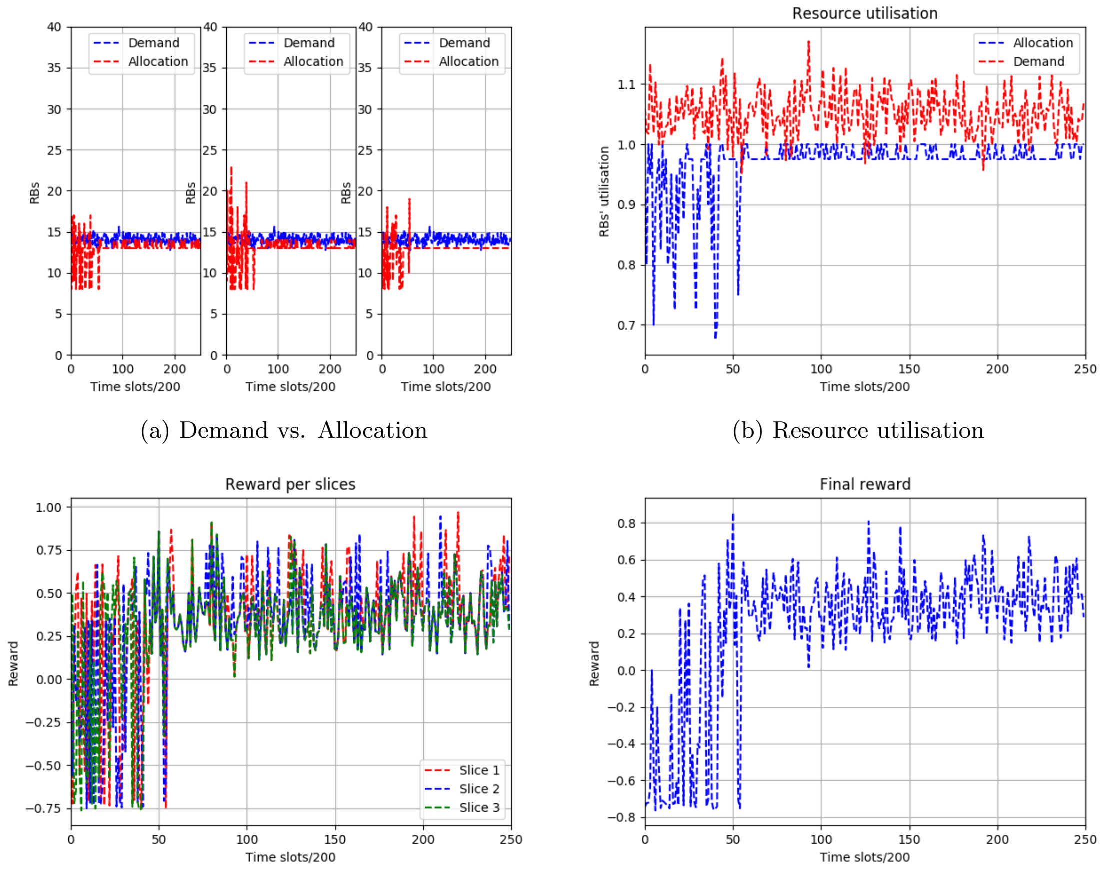
Figure 7.8 – DQN performance for resource allocation task in third scenario

7.4 D-DQN

The D-DQN algorithm was discussed in Section 5.2. The network architecture (Figure 7.5) and parameters set (Table 7.5) for simulations are same as for DQN.

Test 1

Figure 7.9 shows that D-DQN algorithm is still learning noise. However there is a small improvement comparing to DQN, the allocation curve is less fluctuated.



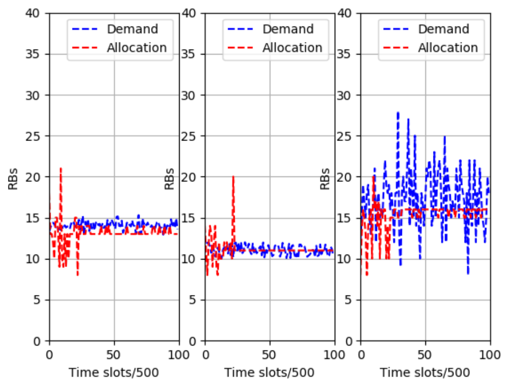
(a) Demand vs. Allocation (b) Resource utilisation

(c) Rewards (d) Final reward

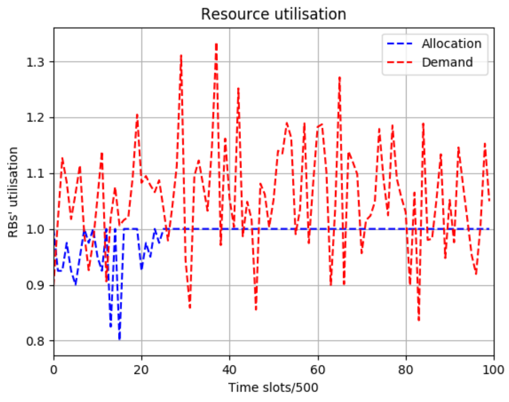
Figure 7.9 – D-DQN performance for resource allocation task in first scenario

Test 2

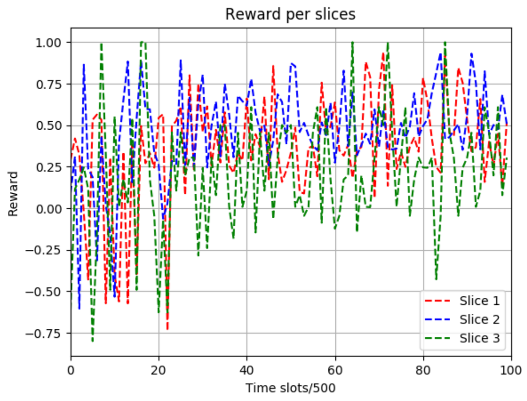
For different IID traffic between slices the D-DQN algorithm converges to a fixed allocation relatively fast. Comparing to a basic DQN technique the learning period is almost two times shorter, from which we can conclude that D-DQN algorithm is better for dynamic resource allocation.



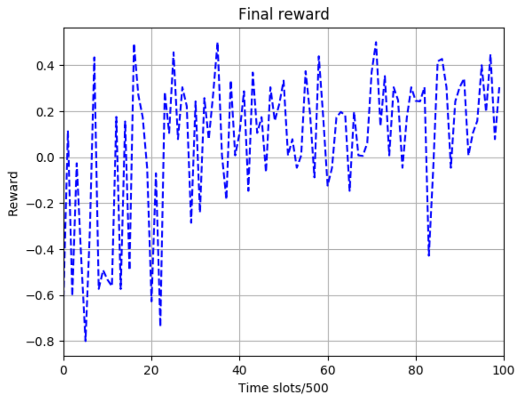
(a) Demand vs. Allocation



(b) Resource utilisation



(c) Rewards

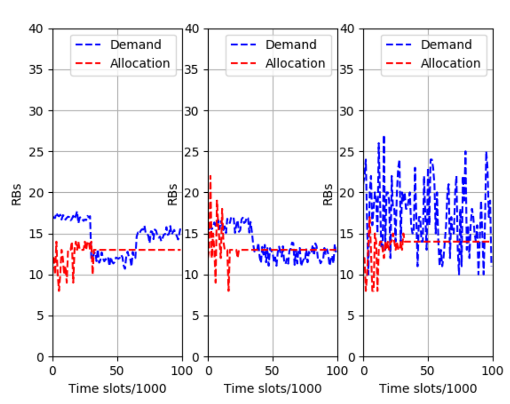


(d) Final reward

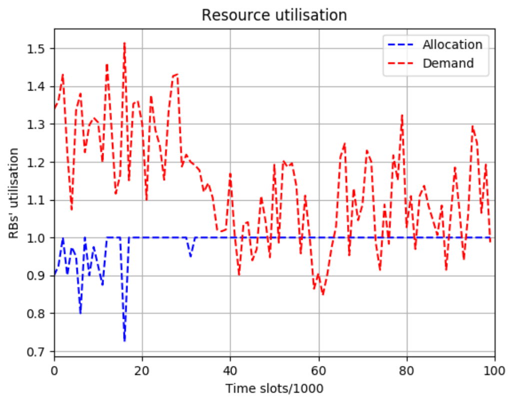
Figure 7.10 – D-DQN performance for resource allocation task in second scenario

Test 3

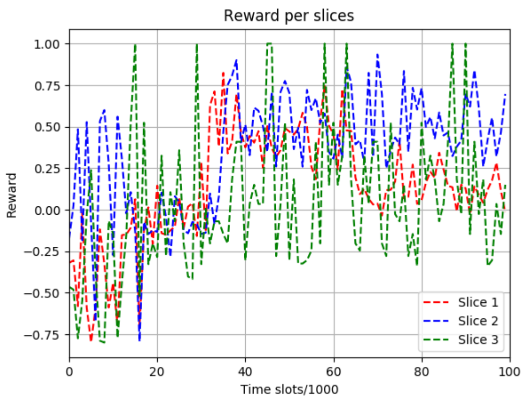
Here we can observe that D-DQN learning based framework adapts to the time-varying environment. Comparing to a basic DQN technique there is a significant improvement in learning period.



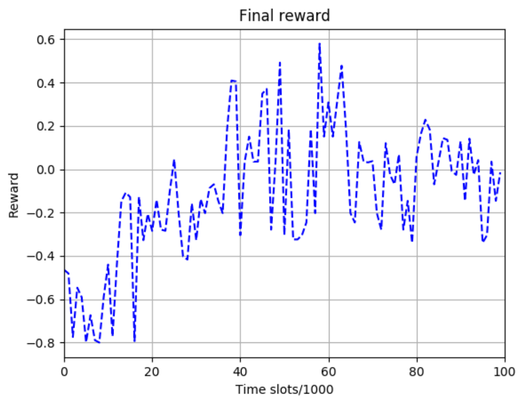
(a) Demand vs. Allocation



(b) Resource utilisation



(c) Rewards



(d) Final reward

Figure 7.11 – D-DQN performance for resource allocation task in third scenario

7.5 Actor-Critic

The AC architecture consists of two neural networks: actor and critic. The structure of the actor-critic DRL agent is depicted in Figure 7.12. The actor is employed to explore a policy π, that maps the agent's observation to the action space. Since the action space is discrete, we use softmax function at the output layer of the actor network so that we can obtain the scores of each actions. The scores sum up to 1 and can be regarded as the probabilities to obtain a good reward by choosing the corresponding actions.

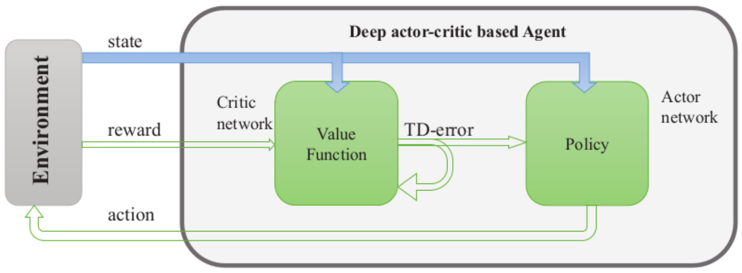


Figure 7.12 – Structure of the actor-critic DRL agent [1]

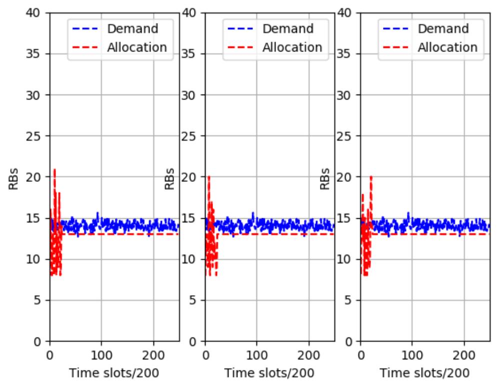
Based on the reward, the current observation space and the observation space for the next time slot, the critic network calculates the TD-error. And finally the critic and actor networks will be updated based on the TD-error. Table 7.2 depicts the parameter setting and DNN architecture of AC agent. The Actor network consists of two hidden layers, while the Critic network contains only one hidden layer.

Table 7.2 – Parameter setting for AC agent

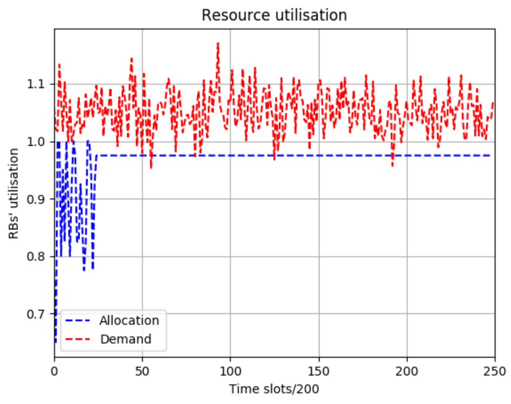
Parameter	**Value**
Actor network	[75, 75] ReLU
Critic network	75, ReLU
Discount factor γ	0.1
Learning rate for actor α_a	0.001
Learning rate for critic α_c	0.01

Test 1

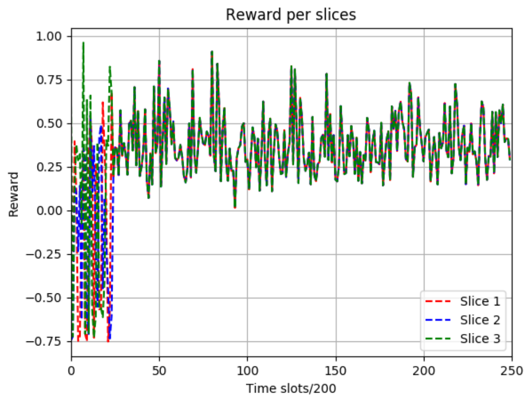
Figure 7.13 proves that AC method has the best performance for IID traffic. The algorithm is not learning noise, it does not have memory. Comparing to other agents, the training period is relatively short. On DRL(c) we can observe that rewards for all three slices are equal, which again confirms the action independence.



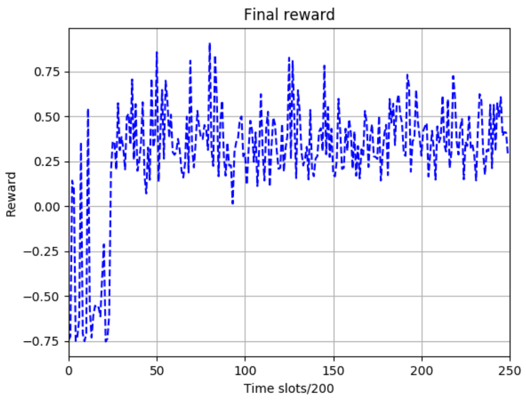
(a) Demand vs. Allocation



(b) Resource utilisation



(c) Rewards

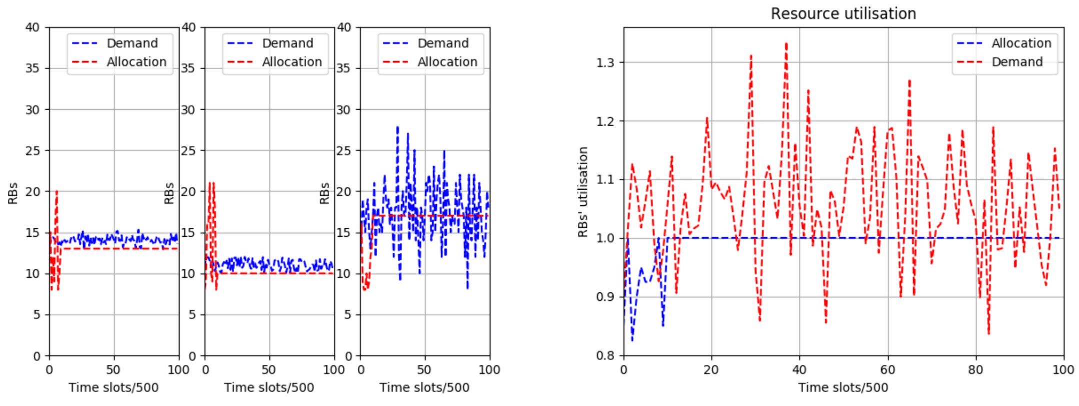


(d) Final reward

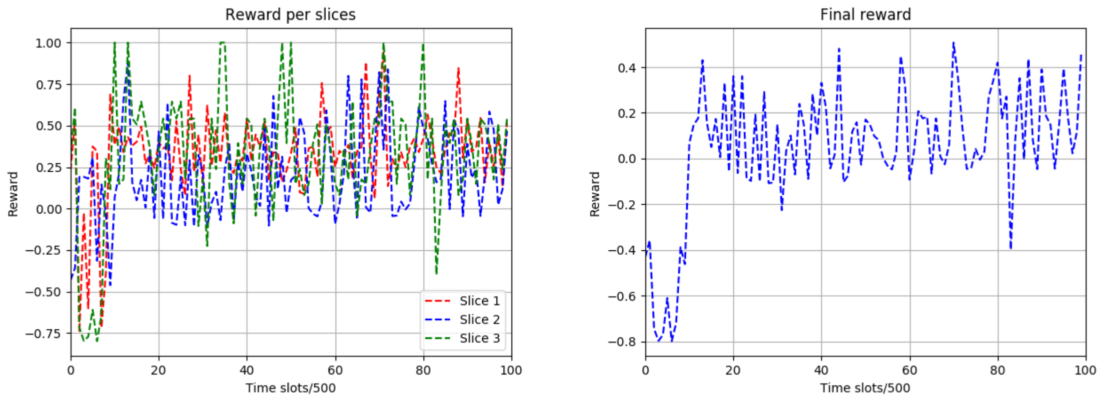
Figure 7.13 – AC performance for resource allocation task in first scenario

Test 2

Here we can see a significant improvement in a learning period comparing to D-DQN. The AC-based algorithm converges to a fixed allocation. If we carefully study Figure 7.14 (a), we can notice that AC algorithm allocates more RBs to slice 3 with a highly time-varying pattern. In a real situation it is preferable to over-allocate traffic to the slice with the most unpredictable demand.



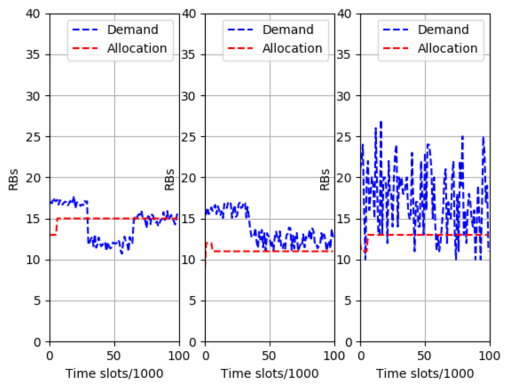
(a) Demand vs. Allocation (b) Resource utilisation



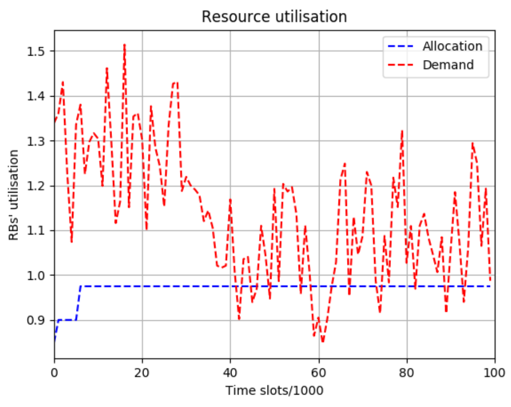
(c) Rewards (d) Final reward

Figure 7.14 – AC performance for resource allocation task in second scenario

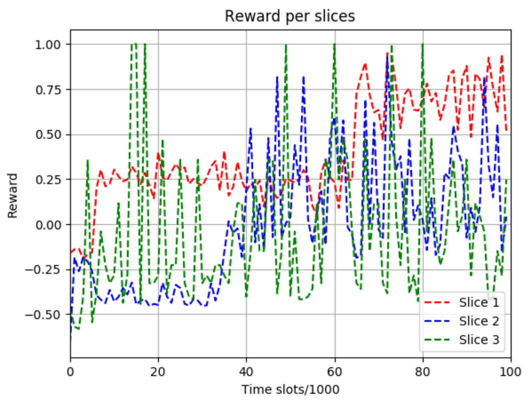
Test 3



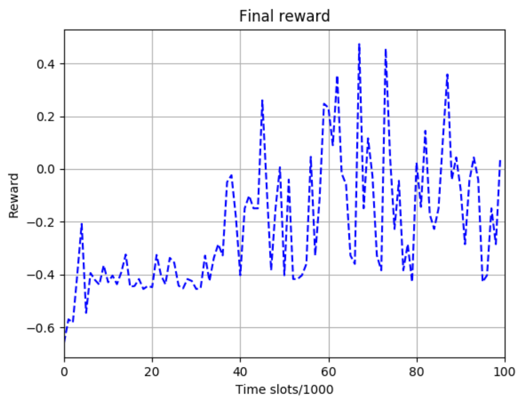
(a) Demand vs. Allocation



(b) Resource utilisation



(c) Rewards



(d) Final reward

Figure 7.15 – AC performance for resource allocation task in third scenario

Figure 7.15 shows that AC-based framework doesn't adapt to the time-varying environment. Comparing to other tested techniques, AC has the worst performance.

7.6 Comparison

In this section we compare proposed frameworks. For each simulation scenario we plot final reward over time and compute average reward over all time steps. In Table 7.3 the architectures of all DNNs and the hyper-parameter settings are listed in detail. The left and right parts of the layer are activation function and neuron number, respectively.

Table 7.3 – Parameter setting and DNN architecture

Parameter	**Q learning**	**DQN**	**D-DQN**	**AC**	
				Actor	**Critic**
Learning rate α	0.01	0.01	0.01	0.001	0.01
Output layer	-	linear	linear	softmax	linear
Hidden layer	-	ReLU 75	ReLU 75	ReLU 75	ReLU 75
		ReLU 75	ReLU 75	ReLU 75	
Input layer	-	linear	linear	linear	linear

The DNN architecture plays important role in learning. High number of hidden layers and neurons can slow down the learning process, which is crucial for dynamic scheduling. On the other hand, agent with a very simple DNN architecture may not converge to the optimal action. In this thesis we design agents NN by running different simulations and evaluating each networks performance.

Test 1

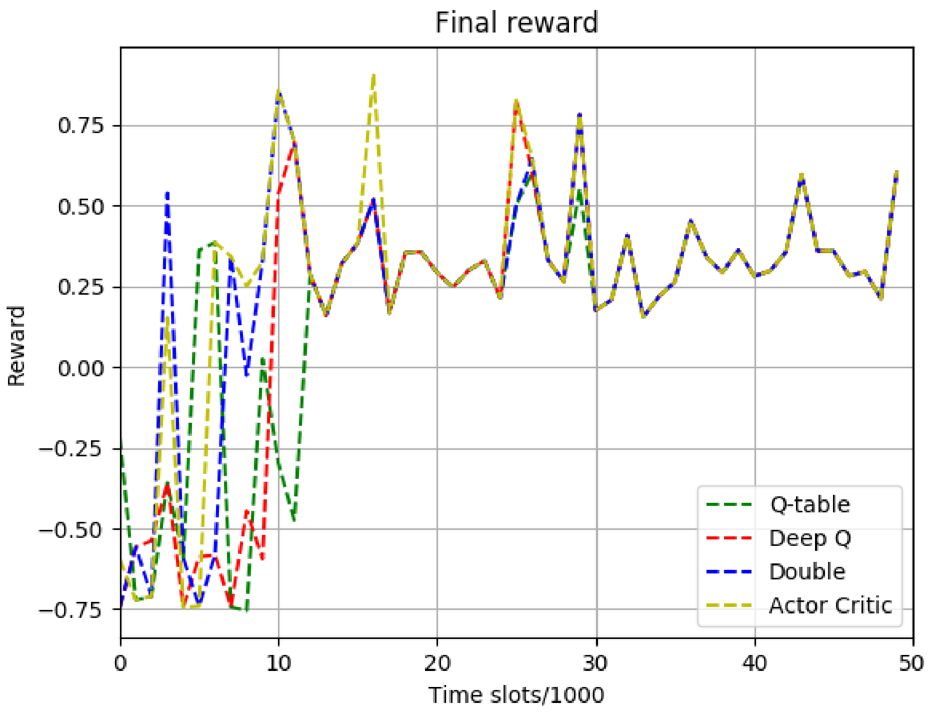


Figure 7.16 – A comparison of the final reward curve of all four trained agents with IID traffic

Table 7.4 – A comparison of the rewards for Test 1

Technique	Average reword over all steps [-]
Q learning	0.180
DQN	0.190
D-DQN	0.245
AC	0.283

In Figure 7.17 we can see that all the agents converge at approximately 30000 timesteps. The best performance shows AC-based framework. The DQN framework does not bring any improvement in the learning period and average reward.

Test 2

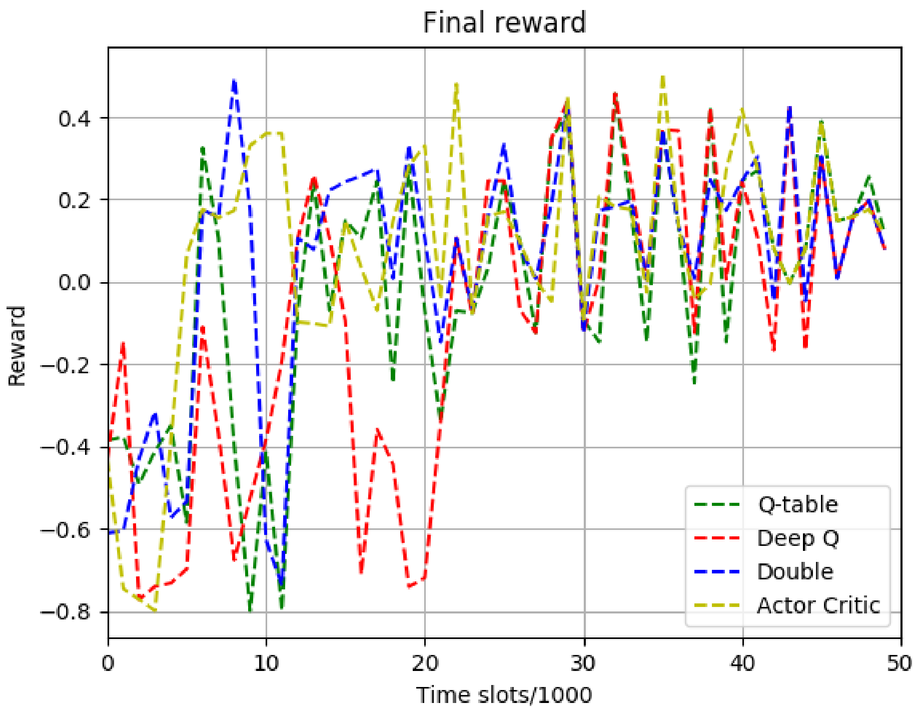


Figure 7.17 – A comparison of the final reward curve of all four trained agents with IID traffic different over slices

We can conclude that AC have the fastest initial increase in reward. It is also interesting to see that DQN does not reduce the variance compared to Q learning and does not bring any improvement in the average reward. D-DQN on the other hand shows quite a big improvement when applying with higher reward and a smaller variance.

Table 7.5 – A comparison of the rewards for Test 2

Technique	**Average reword over all steps [-]**
Q learning	-0.057
DQN	-0.069
D-DQN	0.061
AC	0.043

Test 3

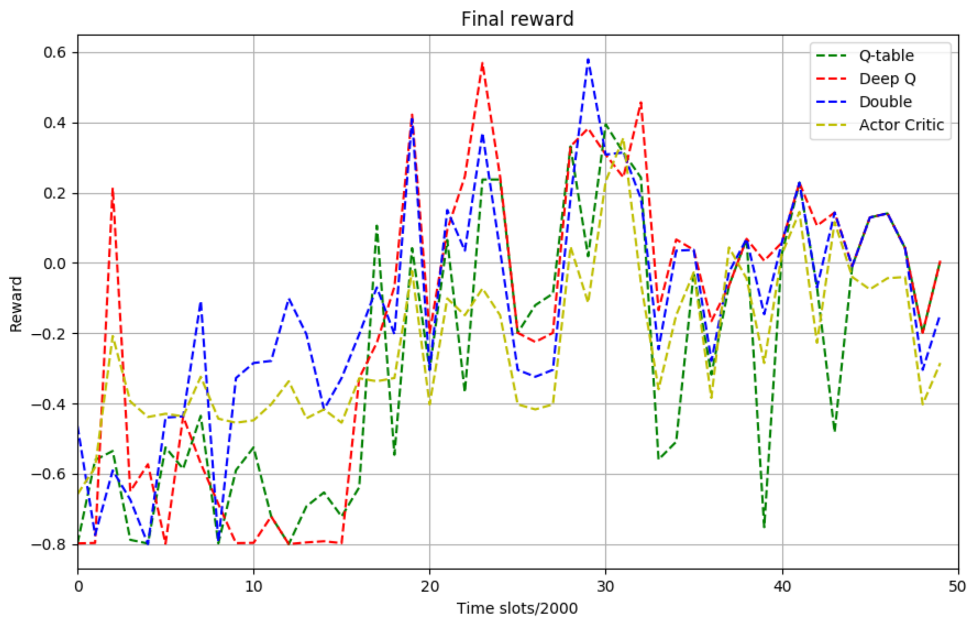


Figure 7.18 – A comparison of the final reward curve of all four trained agents with time-varying traffic

Table 7.6 – A comparison of the rewards for Test 3

Technique	**Average reword over all steps [-]**
Q learning	-0.274
DQN	-0.156
D-DQN	-0.133
AC	-0.239

Based on the average reward in Figure 7.18 and Table 7.6 we can conclude that DQN and D-DQN show very similar results and that they have the highest converged reward among the agents. Both algorithms adapt to the time-varying environment.

8 Conclusion

In this work, we have considered the dynamic resource allocation problem in 5G. To effectively find the scheduling policy, we have proposed and implemented RL and DRL-based frameworks. We have studied four state-of-the-art RL algorithms, based on this, the concrete algorithm design is further developed, and Q learning, DQN, D-DQN and AC algorithms are proposed. Simulations results show that DRL techniques outperform the simple tabular Q learning. However simple DQN algorithm didn't show any improvement in first two simulations. To highlight the adaptive ability, we have run simulations in a time-varying environment and demonstrated that the proposed frameworks learn the new patterns effectively in a relatively short period of time. The AC method shows the best performance for IID traffic resource scheduling, but it does not adapt to the time-varying environment. In the first simulation AC agent gains the highest reward of 0.283, when Q-learning agent has the lowest reward of 0.180. The D-DQN-based technique has the highest reward in most of the cases, for example, for the second and the third simulation the improvement in average reward is equal to 0.118 and 0.141 respectively comparing with Q-learning.

In this thesis we prove that implementation of DRL for resource scheduling is effective and the performance of the network is improved. However, it's not recommended to apply DL techniques in simple tasks. The DRL, is a promising technique for future intelligent networks, and proposed algorithms can be applied to general tasks with discrete/continuous state/action space and joint optimization problems of multiple variables. In particular, the algorithm can be implemented in different networking tasks such as user scheduling, channel management and network planning. Apart from this, there is a room for extension in many possible ways in the future. The DRL algorithms performance can be improved by modifying the DNNs architecture, for example, by using RNNs instead of Feed-forward NNs. The other research direction may be investigation of further DRL algorithms expansion, such as, for example, asynchronous DRL methods.

Bibliography

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, second edition, 2018.

[2] R. Li, Z. Zhao, X. Zhou, G. Ding, Y. Chen, Z. Wang, and H. Zhang. Intelligent 5g: When cellular networks meet artificial intelligence. *IEEE Wireless Communications*, 24(5):175–183, October 2017.

[3] M. Wang, Y. Cui, X. Wang, S. Xiao, and J. Jiang. Machine learning for networking: Workflow, advances and opportunities. *IEEE Network*, 32(2):92–99, March 2018.

[4] C. Zhang, P. Patras, and H. Haddadi. Deep learning in mobile and wireless networking: A survey. *IEEE Communications Surveys Tutorials*, 21(3):2224–2287, thirdquarter 2019.

[5] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y. Liang, and D. I. Kim. Applications of deep reinforcement learning in communications and networking: A survey. *IEEE Communications Surveys Tutorials*, 21(4):3133–3174, Fourthquarter 2019.

[6] Yuxi Li. Deep reinforcement learning. *CoRR*, abs/1810.06339, 2018.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

[8] Q. Mao, F. Hu, and Q. Hao. Deep learning for intelligent wireless networks: A comprehensive survey. *IEEE Communications Surveys Tutorials*, 20(4):2595–2621, Fourthquarter 2018.

[9] Z. M. Fadlullah, F. Tang, B. Mao, N. Kato, O. Akashi, T. Inoue, and K. Mizutani. State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's intelligent network traffic control systems. *IEEE Communications Surveys Tutorials*, 19(4):2432–2455, Fourthquarter 2017.

[10] H. Liu, S. Liu, and K. Zheng. A reinforcement learning-based resource allocation scheme for cloud robotics. *IEEE Access*, 6:17215–17222, 2018.

[11] Xavier Dutreilh, Sergey Kirgizov, Olga Melekhova, Jacques Malenfant, Nicolas Rivierre, and Isis Truck. Using reinforcement learning for autonomic resource allocation in clouds: Towards a fully automated workflow. 05 2011.

[12] Yohan Kim, Sunyong Kim, and Hyuk Lim. Reinforcement learning based resource management for network slicing. *Applied Sciences*, 9:2361, 06 2019.

[13] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–, October 2017.

[14] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[15] Y. Zhang, C. Kang, T. Ma, Y. Teng, and D. Guo. Power allocation in multi-cell networks using deep reinforcement learning. In *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, pages 1–6, Aug 2018.

[16] G. Sun, Z. T. Gebrekidan, G. O. Boateng, D. Ayepah-Mensah, and W. Jiang. Dynamic reservation and deep reinforcement learning based autonomous resource slicing for virtualized radio access networks. *IEEE Access*, 7:45758–45772, 2019.

[17] S. de Bast, R. Torrea-Duran, A. Chiumento, S. Pollin, and H. Gacanin. Deep reinforcement learning for dynamic network slicing in ieee 802.11 networks. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 264–269, April 2019.

[18] R. Li, Z. Zhao, Q. Sun, C. I, C. Yang, X. Chen, M. Zhao, and H. Zhang. Deep reinforcement learning for resource management in network slicing. *IEEE Access*, 6:74429–74441, 2018.

[19] T. Yang, Y. Hu, M. C. Gursoy, A. Schmeink, and R. Mathar. Deep reinforcement learning based resource allocation in low latency edge computing networks. In *2018 15th International Symposium on Wireless Communication Systems (ISWCS)*, pages 1–5, Aug 2018.

[20] H. Ye and G. Y. Li. Deep reinforcement learning for resource allocation in v2v communications. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6, May 2018.

[21] Yasar Nasir and Dongning Guo. Multi-agent deep reinforcement learning for dynamic power allocation in wireless networks. *IEEE Journal on Selected Areas in Communications*, PP:1–1, 08 2019.

[22] Tianshu Chu, Jie Wang, Lara Codecà, and Zhaojian Li. Multi-agent deep reinforcement learning for large-scale traffic signal control. *CoRR*, abs/1903.04527, 2019.

[23] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[24] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 2094–2100. AAAI Press, 2016.

[25] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[26] Matthew J. Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *CoRR*, abs/1507.06527, 2015.

[27] Vijay R. Konda and John N. Tsitsiklis. Actor-critic algorithms. In S. A. Solla, T. K. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 1008–1014. MIT Press, 2000.

[28] Boyuan Yan, Yongli Zhao, Yajie Li, Xiaosong Yu, Jie Zhang, Ying Wang, Longchun Yan, and Sabidur Rahman. Actor-critic-based resource allocation for multi-modal optical networks. *2018 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6, 2018.

[29] Yan Zhang and Michael M. Zavlanos. Distributed off-policy actor-critic reinforcement learning with policy consensus. *CoRR*, abs/1903.09255, 2019.

[30] C. Zhong, Z. Lu, M. C. Gursoy, and S. Velipasalar. A deep actor-critic reinforcement learning framework for dynamic multichannel access. *IEEE Transactions on Cognitive Communications and Networking*, 5(4):1125–1139, Dec 2019.

[31] Y. Wei, F. R. Yu, M. Song, and Z. Han. User scheduling and resource allocation in hetnets with hybrid energy supply: An actor-critic reinforcement learning approach. *IEEE Transactions on Wireless Communications*, 17(1):680–692, Jan 2018.

[32] P. Popovski, K. F. Trillingsgaard, O. Simeone, and G. Durisi. 5g wireless network slicing for embb, urllc, and mmtc: A communication-theoretic view. *IEEE Access*, 6:55765–55779, 2018.

[33] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zhang. Tensorflow: A system for large-scale machine learning. *CoRR*, abs/1605.08695, 2016.

[34] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.