Vrije Universiteit Amsterdam

Universiteit van Amsterdam

Master Thesis

Network Slice Elasticity using Machine Learning

Author: Willemijn Beks (2669090)

1st supervisor: Mw. dr. Chrysa Papagianni
2nd reader: Dhr. dr. Kostas Papagiannopoulos

A thesis submitted in fulfillment of the requirements for the joint UvA-VU Master of Science degree in Computer Science

August 31, 2021

Abstract

With the rise of the 5^{th} generation of mobile communications a need in heterogeneity of network services arises. Network slicing is a vital technology to enable this evolution. Network slice elasticity is one of the core aspects of network slicing and encompasses the horizontal and vertical scaling of network resources. In this thesis we provide some insight into which machine learning methods are viable to use for vertical scaling of network slices in the context of a cellular vehicle-to-network (C-V2N) environment, where cars exchange information with the cloud via Points of Presence (PoPs) that are located throughout the city of Torino. In particular we investigate the use of advantage actor-critic (A2C) method, considering a single PoP. We find that the A2C method is able to generalize well on this problem setting, outperforming previous researched methods. Furthermore, we employ the A2C method on an extended C-V2N environment, where our agent is responsible for vertical scaling as well as assigning each car at the appropriate PoP. Here we get promising results that show that this approach could be successful. We provide an OpenAIGym representation of this environment to encourage facilitate further research.

Acknowledgements

I thank Chrysa Papagianni for all her help and guidance on this thesis. I thank Cyril Hsu for his useful advice and assistance on this project. Thanks to Jorge Martin Perez for answering all my questions about the environment. Thanks to Danny de Vleeschauwer for chairing all the meetings about the project and developing the new environment.

Contents

List of Figures

LIST OF FIGURES

List of Tables

LIST OF TABLES

1

Introduction

The 5^{th} generation (5G) of mobile communications extends the scope of classic mobile communications towards an ecosystem that supports of a wide variety of network services with a diverse set of requirements and capabilities in a single and shared physical infrastructure. The adoption of network functions virtualization (NFV) and software-defined networking (SDN) renders networks more agile. Leveraging NFV/SDN, networks can be sliced logically into multiple, mutually isolated, end-to-end virtual networks composed by a number of customizable software-defined functions, tailored to a given application or service. Thus, network elements and functions can be easily configured and reused in each network slice to meet its respective performance requirements. Automated life-cycle management of network slices poses additional challenges in service orchestration and resource allocation.

The heterogeneity of network demands in 5G mean that the classical approaches are not sufficient anymore, since they offer mostly homogeneous services. Thus the need arises for approaches that improve on the classical algorithms used for problems pertaining to allocation and scaling of network resources. This is where the use of data-driven approaches come in because of their potential for adaptability. The rise of deep learning (DL) in the last decade has opened up possibilities for network management that can handle the ever-increasing amount of data that needs to be processed and managed. DL can be of use by capturing complex dynamics of these high volumes of data, which might be impossible for humans to analyze. Meaning DL solutions could potentially offer us what classical approaches lack.

Lately the research and industrial communities promote the use of ML algorithms for the creation, re-configuration and management of end-to-end slices to fulfill the service requirements, while minimizing the consumption of transport, computing, and storage resources.

Standardization groups, such as the ETSI Zero touch network Service Management (ZSM) and the Experiential Networked Intelligence (ENI), are also working along the same directions. In particular, ETSI ZSM envisions new solutions to realize an agile, fully automated system to support new business opportunities enabled by network slicing, while ETSI ENI focuses on the use of AI for the management of network slices.

Focusing on elastic network slices in the context of 5G, elasticity refers to the mechanisms that can guarantee high service performance and service level agreements (SLAs), when the availability of the computational resources and the traffic loads is stochastic. Thus network slice elasticity can be defined as the ability to gracefully adapt to load and other system changes in an automated way, such that at each point in time the resources allocated to a network slice match the demand as closely and cost and operational efficiently as possible. Elasticity in networks has been traditionally addressed in the context of communication resources i.e., the network gracefully downgrades the quality for all users to deal with shortages in communication resources (bandwidth, spectrum). However, the computational aspects of network slice elasticity are equally important.

Management and orchestration of communication resources as well as resources that are native to the cloud environment (CPU, memory) has become a key challenge in modern telecommunication systems, due to the virtualization and cloudification of networks (**?**). Moreover, more applications will be pushed to the network edge, where the network demands fluctuate heavily. To ensure quality of service (QoS) and to guarantee that SLAs are met, there is a demand for methods that can adequately handle an extensive variety of services in addition to being able to extract relevant information from immense amounts of data. Deep learning can be a worthwhile approach, however it is of vital importance that a suitable method is chosen for a specific problem.

In the context of this thesis, we will investigate a data-driven approach for vertical scaling of computational resources to support communication services in a cellular vehicle-to-network environment (C-V2N). V2N services represent one of the most significant transformations of the automobile industry, gaining increased momentum through 5G networks. V2N services are characterized by having a periodic demand variation over time and by requiring process-intensive and low-latency, reliable computing and communication services. However, as they mostly leverage Edge resources (**?**) as an alternative to Cloud, such more scarce resources must be managed more efficiently.

In the C-V2N environment under consideration, cars exchange information with the cloud via 5G Points of Presence (PoPs) that are located throughout a city. Whenever a car enters into the neighborhood served by a PoP, its workload needs to be supported by

the computational resources of that PoP. It is the job of the resource controller to determine that number of CPUs, as cars travel through the city, and hence, as the workload that is presented to that PoP fluctuates over time. Reinforcement learning (RL) can be beneficial in these types of problems since it is able to learn online and adapt to environment with high fluctuations, like a typical 5G network environment.

Towards that end, we investigate the use of advantage actor critic (A2C) RL approach to optimise network slice elasticity in the context of C2N services and compare this approach with the existing research of **?** (**?**). A comparison of these methods will provide insight in the way forward for data-driven network slice elasticity. Data-driven approaches can outperform classical approaches in network management (**?**). Thus we propose to shed some light onto the question, *which type of data-driven approach can be successfully applied to solve problems in automatic network management.* The experimental results show that our chosen data-driven approach of A2C methods can outperform the LSTM, which was the optimal approach that was experimented with(**?**). The A2C is superior to their Q-learning approach as well, because as a policy-gradient method, A2C utilises a policy function as well as a value-function, i.e. it can learn a broader range of information in comparison to a Q-learning approach. Our preliminary results in an environment with more agents and the ability for the agent to also place cars, our A2C approach also shows promising results. The main contributions of this thesis are i) an A2C implementation that can be applied to optimise network slice elasticity in the particular context ii) comparison and analysis of the A2C approach with previous research results from (**?**) and iii) an application of the A2C method on an extended version of the problem (and environment) presented in (**?**); to this end we make a wrapper for the new environment in the OpenAIGym (**?**) format, enabling future research on this environment with various other ML techniques.

2

Background

2.1 Network slice elasticity

With the advent of 5G systems as the new pivotal infrastructure, profound changes are taking place in how we design and implement cellular networks. 5G encompasses several technologies that provide possibilities for differentiation on service, thus enabling support of a diverse set of use cases with different requirements (**?**). Such new and cost-effective technologies that facilitate the necessary network adaptability, such as network slicing, Mobile Edge Computing (MEC) etc, are of vital importance. One of the many use cases 5G has to offer is related to the automotive industry. Modern vehicles often require efficient networking capabilities to sustain IoT functionalities as well as edge-cloud applications. These technologies enable new functionalities such as autonomous driving or real-time assessment of certain traffic conditions.

Network slicing as a concept in 5G networks can be defined as the process of having multiple self-contained logical networks on top of a shared physical infrastructure (**?**). A network slice can encompass all aspects of the network and might contain an entirely different set of network functions than another network slice. The aim of a network slice is to provide appropriate traffic processing based on its specific use-case. As a technology it enables different customized networks for multiple use-cases, that differ in terms of functionality, performance, and isolation, while using the same hardware (**?**). Thus network slicing is able to provide a customized network operation and service differentiation. It is a valuable technology by virtue of its ability to offer a wide range of resources (e.g. radio, networking, cloud) to parties that have a need for a customized network operation (**?**).

The successful existence of network slicing is facilitated with the rise of network and IT virtualization technologies and network programability. To understand network slicing, it

is vital to understand the key enablers of this technology concept.

Network virtualization can be defined as creating a representation in software for available resources for the data and control-plane functions (**?**). In network virtualization there is a decoupling of functionalities into infrastructure provision and service provision. Infrastructure provision entails managing of the physical hardware, while service provision entails providing end-to-end network services (**?**). In *Network Function Virtualization* (NFV), network textit are implemented in software instead of proprietary hardware. Consequently, this allows for more flexibility with the implementations of said network functions as well as less dependency on hardware-specifics. Network functions can be deployed in VMs or container on commodity servers. This NFV framework consists of three main parts. Network functions that are implemented in software are virtualized network functions (VNF), the *NFV infrastructure* (NFVI) in both hardware and software, and the entity that is responsible for managing and orchestrating (MANO) the VNFs (**?**).

Software-Defined Networking (SDN) is a network architecture approach that enables the network to be intelligently and centrally controlled (programmed), using software applications. In essence the data-plane is decoupled from the control-plane, so that forwarding decisions can be made in a centralized fashion and are programmable. This enables higher flexibility, programmability, scalability and less hardware dependability than in classic networking management (**?**).

Cloud and *edge computing* are the physical enablers of network slicing by contributing physical resources like storage, networking, or computational resources (**?**). *Cloud computing* builds upon advances on virtualization and distributed computing to support cost-efficient on demand usage of remote computing resources, emphasizing on elastic scaling and establishing a pay-per- usage business model for IT services. With virtualization, a separate layer of abstraction is added between physical infrastructure and the operating system. This layer is called a hypervisor and can be defined as firmware that is responsible for virtual machine (VM) management. A hypervisor provides a virtual interface for external operating systems that seek to execute applications or other services. Network slice instances can share hardware resources because of hypervisors overseeing the process (**?**). A virtual machine (VM) is the virtualization of a physical machine that runs its own OS. A VM emulates an OS, but it is fully isolated from the actual hardware. A container is an isolation on a smaller scale than a whole OS, more often used to package a single application. Running a network slice on a VM offers full isolation, but using a container provide a more light-weight solution.

MEC aims to provide an IT service environment and cloud-computing capabilities at the edge of the mobile network. MEC proposes co-locating computing and storage resources and executing applications at multiple locations at the edge of the mobile network (e.g.,5G base stations etc.), enabling computing applications, data management and analytics, as well as service acquisition, assuring data proximity, ultra-low latency and high data rates.

There are seven main principles that are the basis for network slicing. These are automation, isolation, customization, programmability, end-to-end, hierarchical abstraction, and elasticity (**?**). We will elaborate on elasticity since it is vital to the topic of this thesis.

Elasticity refers to the adaptation of the resources that are assigned to a certain network slice, with the objective of meeting the service-level agreement (SLA) of that specific slice. This adaptation should happen under fluctuating radio and network conditions, amount of users in the system, and geographical area of the users (**?**). Resource elasticity can be achieved by scaling horizontally or vertically, or by modifying virtual of physical network functions in the slice. In the context of network management there are two types of scaling. One can scale in/out or up/down, horizontal or vertical respectively. Scaling horizontally is defined as adding more resources to the existing resource pool. This can mean using more machines. Scaling vertically is defined as adding more resources to an already active machine, e.g. more RAM, GPU or CPU. Note that for some of these actions there needs to be negotiation with other slices in the network because they might be competing for resources if a slice requires a finite resource.

2.2 Reinforcement learning

Reinforcement learning (RL) is a type of machine learning where a self-learning agent can interact and learn from its environment. RL is distinct from the other two other main types of machine learning, which are supervised and unsupervised learning. The defining aspects of RL are the agent that can interact with its environment. Furthermore, there is (i) a reward function, which is numerical feedback from the environment to the agent and (ii) a policy, which defines the agents behaviour, or more specifically which actions to take in certain states of the environment. A policy can be stochastic or deterministic and are at the core of the reinforcement learning agent because it is sufficient to define behaviour (**?**) Additionally, there is a value function, which predicts the long-term success of a chosen action. Meaning that it takes into account the probability of possible rewards in future states. A value of a state is the total expected reward of the current state and the ones that are probable to follow.

On every (time)step, the agent chooses an action and consequently receives feedback from the environment. This feedback is either a positive or a negative reward which corresponds to how optimal the chosen action was. After receiving the award the agent updates its policy to reflect this feedback.

RL is most useful in a setting where there is some policy to optimize and we want our model to keep improving while we use it. In RL there is no explicit divide between right or wrong decisions, there is learning an optimal mapping from the state of the environment to an action that will produce the highest reward.

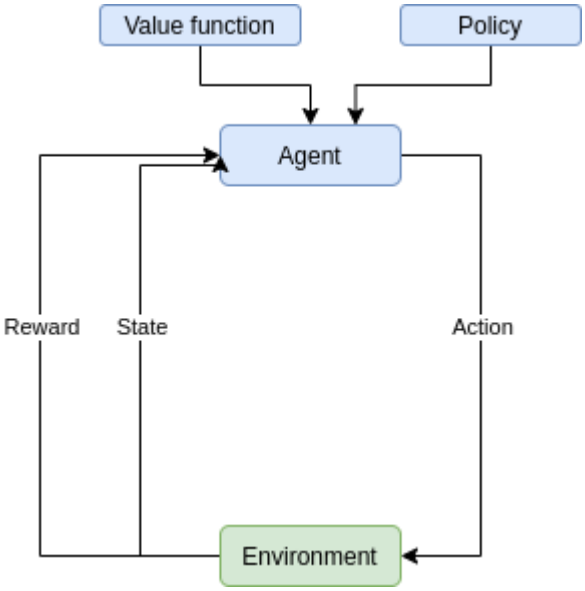

Figure 2.1: Basic overview of reinforcement learning.

2.2.1 Important concepts in reinforcement learning

In this section we will briefly introduce some concepts that are vital to understanding the research in this thesis.

2.2.2 Return

A RL agent aims to maximize total rewards over time, meaning the agent seeks to maximise *expected returns*. The returns are a function of the rewards, often formally defined as $G_t = R_{t+1} + R_{t+2} + ... + R_T$, where T is the maximum time-step and G_t is the return at time-step t (**?**). Contingent upon this definition is that there is some final time-step T, which is not always the case.

2.2.3 Episodic and continuous learning

If there is a logically definable end to a sequence of interactions between an agent and an environment, one such a sequence is called an *episode*. Intuitively this could be one single event like one single game of chess, or managing a car-ride from start to destination. An episode ends at its terminal state, after which the environment should be reset to its initial state. It is of importance to note that this means that each episode is independent of the previous episodes, in the sense that it does not matter what the outcome of any previous episodes were because the starting state is unchanged. It is not a given that a task is episodic in nature, because many interactions between agents and environments do not have a natural ending.

Calculating the returns for continuing tasks is not straightforward. Mainly because a possibly infinite sequence, means that the final time-step might be at time infinite. To combat this problem, discounting with parameter γ is introduced. With discounting factor introduced, the returns function is expressed as following: $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} y^k R_{t+k+1}$, where the discount rate *gamma* is $0 \leq \gamma \leq 1$ (**?**). A discount rate close to one means that the agent will be interested in rewards far in the future, meaning it cares about its long-term strategy. While a discount rate closer to zero implicates an agent that cares more about rewards in the immediate future. Note that for an episodic task, γ can be in the range of $[0, 1]$, but for a continual task, γ should be in the range of $[0, 1)$.

2.2.4 Exploration and exploitation

In RL there needs to be a balance between *exploration and exploitation*. This means that there needs to be a balance between maximizing immediate rewards, and exploring new strategies, that possibly lead to even better long-term rewards. An agent in RL does not possess complete knowledge of its environment, otherwise there would be no need for learning. So it is conceivable that a strategy that is learned and seems to be optimal at the time, is actual a local minimum or maximum. So if we use little to no exploration, an agent might get stuck in a sub-optimal strategy. Whereas too much exploration can lead to high variance and thus difficulty to converge. The exploration and exploitation trade-off is an important part of learning in RL.

Among other strategies, adding e.g., a *temperature* to the softmax activation function is a way to stimulate exploration in a network. The softmax function with temperature is

described as following:

$$\sigma_i = \frac{e^{\frac{y_i}{T}}}{\sum_{k=1}^{n} e^{\frac{y_k}{T}}}$$

where y is the input vector and T represents the temperature. The value of the temperature should be a positive number. When the temperature T gets larger, the effect is that the predicted probabilities are closer together than when the temperature T has a value closer to zero. With a higher temperature, the model gets less secure about its predictions than before. This is desirable behaviour if one wants to increase exploration. Higher temperature means more exploration, while no temperature means no exploration.

Another strategy to improve exploration is entropy regularization. Entropy regularization can smooth the optimization landscape, improving learning capabilities of a network(**?**). Entropy regularization is done by calculating the entropy of the distribution that was produced by the forward pass of the policy network. Entropy refers to how unpredictable the actions of an agent are, or in terms of the policy, how certain the predicted distribution is. High certainty entails low entropy, while low certainty entails high entropy. The entropy function is described as following:

$$H(x) = -\sum_{i=1}^{n} P(x_i) log_e P(x_i)$$

where x is the input distribution. After each episode ends, the entropy is combined with the loss with a scaling factor k. The scaling factor k can be tuned to optimize a specific problem, as its optimal value varies. The effect of the entropy regularization is that it only advocates for strong decisions when the network is very certain about its decisions, otherwise it redistributes the probabilities to get a more evenly allocation of values. The consequence of this is that it enables an agent to learn a more diverse policy, that can ultimately lead to a more optimal policy.

2.2.5 Temporal difference and Monte Carlo

There are several ways to learn in RL, there is Monte Carlo methods and temporal difference (TD) learning. Both methods utilise experience that they acquired by following some policy π to learn to improve predictions. Given this experience, both methods update their estimate of v_π for non-terminal states. However, Monte Carlo only makes estimates at the end of an episode, while TD-learning can do it as often as every step. The main difference between Monte Carlo and TD-learning is that TD-learning uses bootstrapping to approximate the action-value function and Monte Carlo uses an average to estimate

the action-value. To accomplish the frequent updates, TD-learning uses bootstrapping to make an estimate of the value function v_π. This means that TD-learning is susceptible to bias, especially at the start of learning when these estimates might be very far from the correct values. Since Monte Carlo only updates after an episode, it can use actual values for its estimates. Monte Carlo uses an average to estimate the value function v_π. However, this method can cause high variance. To combat this, a high number of samples has to be utilised before reaching a sufficient level of learning in comparison to TD-learning.
To elaborate on how TD-learning accomplishes its bootstrapping methods, the TD-error is explained. The temporal difference error (TD-error) measures the difference between the estimated value of the (non-terminal) state and the more optimal estimate that can be made one time-step further. Formally the TD-error is defined as $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$, where V is the value function of state S_t is the state at time-step t (**?**).

2.2.6 Value methods and policy gradient methods

Reinforcement learning algorithms can be classified into two categories: action-value methods and policy gradient methods. Both methods have their individual trade-offs, which will be discussed in this section. Actor-critic methods utilize both methods and thereby utilizes the respective advantages of value methods and policy gradient methods. With value methods, the values of state-action pairs are learned and actions are subsequently selected based on their estimated values and the current state. There are multiple action-value methods, of which we will introduce Q-learning since it is relevant to this thesis. Q-learning is an off-policy TD control algorithm that was one of the first successful RL algorithms (**?**). Independent from the policy, Q-learning directly approximates the optimal action-value function q_*. All action-values need to be continuously updated with every interaction with the environment. Q-learning is defined by $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma\ max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$, where Q is the action-value function, S is the state, A is the action, R is the reward, t is the time, α is the step-size, and γ is the discount factor.

While with policy gradient methods, the goal is to learn a parameterized policy that can be used to select appropriate actions. Policy gradient methods seek to maximize a scalar performance measure $J(\theta)$ with respect to the policy parameter θ. The updates are approximated by applying gradient ascent to calculate $\theta_{t+1} = \theta_t + \alpha\nabla J(\theta_t)$ for time t. Note that the stochastic estimate $\nabla J(\theta_t) \in \mathbb{R}^d$ which expected value is an approximation of the gradient of the performance measure with respect to θ_t and d represents the dimentionality (**?**). There are multiple approaches for defining the policy with policy gradient methods,

as long as the policy $\pi(a|s, \theta)$ is differentiable with respect to its parameters. Often it is good to require the policy to not be deterministic as to ensure possibility of exploration (**?**).

Policy gradient methods can learn probabilities for choosing certain actions, as well as utilizing an appropriate level of exploration to find an optimal policy and prevent local minima. These qualities are either absent or significantly harder to implement in value-based methods. Depending on the specific problem, it might also be more straightforward to define the policy than the value function parametrically. However, selecting an appropriate way to evaluate the quality of the policy is not always straightforward in policy gradient methods, which can be a possible downside. An additional advantage of policy gradient methods is that there is a solid theoretical foundation behind it. The policy gradient theorem provides a theoretical framework to calculate policy parameter influence on performance in a way that is not dependent in any way on derivatives of the state distribution (**?**). This means that the unknown effect of policy changes is not needed.

2.2.7 Actor critic methods

Actor critic methods are a type of RL methods that combine policy gradient learning and action-value learning. There is a policy gradient section of the actor critic method that is concerned with selecting actions, this is the actor. The other section of the critic method is concerned with providing numerical feedback to a chosen action, this is the critic.

Actor critic methods combine the advantages of policy gradient learning with the advantages of action-value learning.

In actor critic methods, an agent transitions between states and receives a reward for each action equivalent to other RL methods. However, there are two outputs generated by actor critic methods, namely the expected reward i.e. the state or action value and the policy distribution for the possible actions. As can be seen in figure 2.2, the actor chooses the action while the critic provides feedback about the quality of the action. Actor critic methods can be implemented with TD learning as well as Monte Carlo.

The particular variation of actor-critic learning that is relevant for this thesis is advantage actor-critic (A2C) learning. To combat high variability in regular actor-critic learning, A2C uses the 'advantage' function to stabilise. The advantage function can be understood as a measure for how much better a particular action at a particular state is, compared to the expected value for an action at that state. If the advantage function returns a positive value, the chosen action was better than expected, and if the returned value is negative, the chosen action was worse than the expected value. The advantage function can be defined as

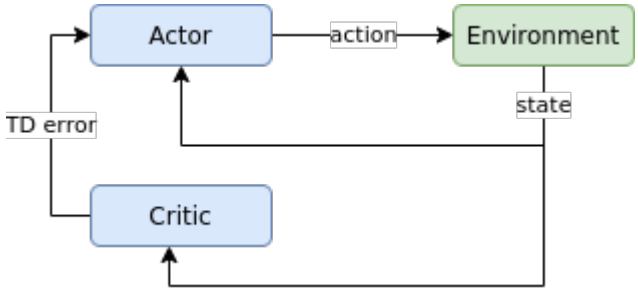

Figure 2.2: Basic overview of actor-critic learning.

$A(s_t, s_t) = Q_w(s_t, a_t) - V_v(s_t)$. Since this formula consists of two different value functions, two networks would be necessary to calculate the advantage function. Fortunately, because $Q_w(s_t, a_t) \ = \mathbb{E}[r_{t+1} + \gamma V(s_{t+1})]$, Q can be estimated with the TD target $r + \gamma V(s'_t)$. In the Advantage Actor Critic algorithm, the Advantage is equal to the TD Error, meaning $A(s_t, a_t) = r_{t+1} + \gamma V(s_t + 1) - V(s_t)$.

Below is pseudo-code for Monte Carlo state-value A2C:

Algorithm 1 Advantage actor critic (A2C)

Initialise actor network (π_θ), critic network(V_w), learning rates α_θ and α_w
for episode in total number of episodes **do**
    state s = reset initial state
    while episode not done **do**
        sample action distribution from actor $a \sim \pi_\theta(a|s)$
        calculate state value from critic $V_w(s)$
        take action with actor and get new state s' and reward r
        $s \leftarrow s'$
    returns = rewards +γ* discounted sum
    advantage = returns - values
    loss_actor = log(-action_probabilities * advantage)
    loss_critic = MSE(state values, returns)
    update actor π_θ and critic V_w

3

Related work

This chapter introduces related research in the field of ML-based resource management in networks. At first, research that utilises RL to handle research management for network slices is discussed. In the following we focus on research that applies machine learning to scaling problems on network slices.

3.1 Resource Management

? look into the benefits of using RL in network slice resource management by comparing existing methods to deep RL methods. Effectively matching allocated resources to slices according to their activity demands is a vital part of effective network slicing (**?**). They show the effectiveness of deep Q-learning by simulating radio resource slicing and priority-based core network slicing. **?** mention that DRL methods show promising results by probably learning deeper implicit relationships between user-demand and resource-supply. The authors mention that deep RL could play a vital role in network slicing in the future. The authors identify the following four prominent factors for successful network slicing. Firstly, a highly capable slice management system. Secondly, choosing an appropriate abstraction of actions and state. Thirdly, latency and accuracy for reward retrieval should be taken into account. Fourthly, policy learning can be time intensive, while large variations in wireless channel traffic causes a demand for faster learning than their current implementations can provide. Combining RL with other ML methods might even further advance successful resource management for network services. There are several papers that find that their approach is highly effective. **?** look into a combination of A2C and LSTM methods to improve automated resource management in network slicing taking into consideration user mobility (**?**). They use a state-processing LSTM network that acts as

input to an A2C network that aims to optimise an access network consisting of multiple base-stations. The goal is to maximise a utility function based on spectrum efficiency and SLA satisfaction level, while at all times guaranteeing SLA satisfaction for all slices. In their problem definition, the state is the number of packets that are arriving at a time, while the action as the amount of bandwidth that is assigned to each network slice. **?** find that their proposed A2C-LSTM solution is able to effectively guarantee SLAs as well as improve spectrum efficiency.

In an attempt to optimise resource scheduling for 5G radio access network slicing, **?** utilise both an LSTM as well as asynchronous actor-critic (A3C). The main target is to effectively guarantee a sufficient degree of performance isolation between slices and at the same time improve multiplexing gains. They intend to utilise the best of A3C's online learning capabilities, along with LSTM's capability of accurately catching broad-scale traffic patterns (**?**). The authors do this by employing the LSTM to perform large time-scale resource allocation, while getting the A3C to perform on-line resource scheduling. The authors find that they achieve significant performance increase when comparing them to benchmark algorithms.

? combine a deep distributional Q-learning network (DDQN) and a GAN to optimise demand-aware resource management in network slicing (**?**). They look into a scenario where several base-stations that share the same physical resources are part of a radio access network with several network slices. They model the problem by taking the services demands as the state, and the resources that are allocated as the action. The main contribution of the paper is the design of the GAN-DDQN and using the GAN to generate an action-value distribution and thereby circumventing the influence of randomness and noise on the attainable reward of the RL. Their GAN-DDQN model outperforms DQN algorithm in calculating the SE and SLA satisfaction ratio (SSR).

3.2 Resource Scaling

Employing machine learning to solve scaling of VNFs is an active field of research that relates closely to scaling problems in network slices since it is also concerned with predicting network traffic patterns and modifying behaviour accordingly.

? experiment with horizontal auto-scaling of VNFs with various ML methods for optimizing cost as well as QoS guarantees (**?**). They experimented with several supervised learning ML classifiers and find that the random forest classifier is the most accurate one, as it can predict how many VNF instances are necessary at a moment in time with 96%

accuracy. **?** are distinct from previous studies on horizontal auto-scaling of VNFs because they try to make proactive decisions instead of reactive ones.

Making scaling decisions pro-actively by utilising neural networks to decide the optimal amount of VNF instances needed, can be done with a success rate of 97% on traffic traces of a commercial mobile network (**?**). **?** use a straightforward multi-layer perceptron (MLP) to learn a function that maps traffic load statistics to an appropriate number of VNFs. Their features include traffic load as well as several constructed features that enable a standard MLP to take some time-series information into account.

? compare ML methods with classical algorithms as well as a static approach and test them on a service scaling prototype that automates vertical scaling decisions in (**?**). The prototype aims to automate scaling decisions that need to be made to meet SLAs as well as minimize resources being used. They use a constant CPU factor, a classical controller, Q-learning reinforcement learning, and an long short-term memory (LSTM) network. The LSTM had the most optimal results from the ML methods that were tested. Their findings show that the LSTM was the most proficient at predicting workload that was based on traffic flow that comes into the range of a certain PoP. These findings coincide with that of (**?**). **?** have shown that LSTM networks are sufficiently capable in predicting short-term (car) traffic in cities.

Overview related research			
Paper	**Subject**	**Scaling type**	**Methods**
? **?**	Resource Scaling	Vertical	LSTM, RL, PI
? **?**	Resource Management	n.a.	DDQN, GAN
? **?**	Resource Management	n.a.	Q-learning
? **?**	Resource Management	n.a.	A2C, LSTM
? **?**	Resource Scaling	Horizontal	ML classifiers
? **?**	Resource Scaling	Horizontal	MLP
? **?**	Resource Management	n.a.	A3C, LSTM

Table 3.1: Overview of related work to this thesis

4

Method

To get a better understanding towards the research question of which data-driven approaches can successfully be applied to the problem at hand, we implement an A2C network and test it in different settings.

4.1 A2C

4.1.1 Implementation

Our A2C is implemented in PyTorch (**?**). It consists of a separate actor network as well as a critic network. In some learning scenarios it can be beneficial to utilise multi-task learning (**?**). In multi-task learning multiple networks share some initial layers with the goal of improving learning. This can be done because earlier layers in neural networks are known to contain more general information about the problem. We have also implemented a version where the actor and critic share a part of the network architecture, however we found no difference in performance for our current applications so we have not used it to acquire or final results.

We use the exponential linear unit (ELU) activation function, since ELU facilitates faster learning and better results on common benchmark datasets (**?**). ELU behaves in the following way with x as its input value:

$$h(x) = \begin{cases} x, & \text{when } x > 0 \\ exp(x) - 1, & x \leq 0 \end{cases}$$

which results in the following behaviour when updating:

$$\frac{\partial h}{\partial x} = \begin{cases} 1, & \text{when } x > 0 \\ exp(x), & x \leq 0 \end{cases}$$

We use the adaptive moment estimation (Adam) Optimizer (**?**) to train the A2C networks. Adam is a replacement for stochastic gradient descent (SGD) that only requires first-order gradients. Adam uses a separate learning rate for each network weight to decrease relative distance between the weights. Adam combines the advantages of RMSprop and AdaGrad (**?**). We use a learning rate of 0.0001. The discount factor is $\gamma = 0.998$. The entropy loss factor is set to 0.01. We use Monte Carlo methods to train our network.

4.1.2 Benchmarking

To validate the A2C network, it is trained on the CartPole-v0 environment from OpenAI Gym (**?**). If the A2C network is able to learn how to hold the cart-pole upright, we can conclude the network is able to learn and is thus implemented correctly. The A2C network is considered to have successfully addressed the the cart-pole balancing problem if it is able to get an average reward of 195.0 for over 100 consecutive trials (**?**).

4.2 Data-driven ML-based Scaling for V2N Services

We extend the research in (**?**), by adding another data-driven method to solve the proposed problem. As described in section 3, **?** apply four methods to the problem of auto-scaling (5G) V2N cellular services (LSTM, Q-learning, a constant factor, and a Proportional Integral controller). In this section, first we will explain the methods that are relevant to this thesis. Second, we will explain the Torino environment and corresponding dataset. And third, we will explain our experiments on this environment.

4.2.1 Comparison Method

We compare the A2C results with the LSTM network and the applied RL approach (Q-learning) as described in (**?**). The LSTM outperformed the other methods in the experiments. Q-learning is included as a directly comparable RL method.

An LSTM network is a long short-term memory network that is able to store temporal information (**?**). LSTMs are a special case of recurrent neural networks (RNN), that are able to learn long-term dependencies in data. They are one of the most successful models that are used in prediction of long-term time-series forecasting. The defining idea behind LSTMs is the cell state, that enables information to be passed on from step to step.

An LSTM consists of several parts. The first step in an LSTM is the forget gate layer, which is a sigmoid layer that decides how much information of the previous state is kept. Then there is the input gate layer, which decides which functions should be updated. Next,

there is a tanh layer that creates the values that are to be added to the state. This is all combined into a regulated output. Some variations are possible, although this remains the basic structure. LSTMs can be trained with time-series data and can learn patterns of traffic flows (**?**).

The second method from the paper we are comparing against is Q-learning RL. Q-learning is an off-policy TD model-free RL algorithm (**?**). The algorithm learns by continuously evaluating and improving the value of an action in a given state. The algorithm is considered model-free since it does not use a model to consider possible future situations before they are experienced. The goal of Q-learning is to learn a policy, which tells an agent what action to take (**?**). Intuitively, Q-learning works by having the agent select the actions that are most likely to lead to the optimal long-term reward. As it experiments more, the agent gains more knowledge, and its long-term strategy improves. If all state-action pairs are visited during the learning process, then Q-learning is guaranteed to converge to the correct Q-values with probability one (**?**).

4.2.2 Torino dataset and environment

The environment where we compare our results with those of **?** supports cellular vehicle-to-network (C-V2N) communications where cars exchange information with the cloud via PoPs that are located throughout the city of Torino (**?**). When a car enters the area that is under control served by a PoP, its workload needs to be supported by the computational resources of that PoP. Computational resources are available in discrete chunks expressed as the number N of CPUs activated at a certain moment in time. It is the job of the controller at each PoP to determine that number of CPUs, as cars enter the area under its control, and hence, as the workload that is presented to that PoP fluctuates over time. In other words the controller at the PoP has the ability to scale vertically according to the new workload. It can choose to increase, decrease CPU resources, of remain in its current state.

The dataset that is used in the research of **?** is based on traffic patterns in the city of Torino in 2020[1]. The trace contains over 100 measuring points, from January 2020 to October 2020, counting the number of cars that passed certain measuring points over consecutive intervals of 5 minutes, plus add In the paper authors consider a single one that reflects the PoP at street Corso Orbassano.

[1] http://opendata.5t.torino.it/get_fdt/get_traffic_data

In this environment, thus the agent (controller) makes decisions for a single PoP. The input from the environment to the agent consists of the required workload for the incoming cars for every timestep t. There is a CPU range wherein the agent can scale up or down. If the number of active CPUs is not sufficient to support the workload of the cars in the area, a penalty is imposed to account for this. On the other hand, overprovisioning CPUs, meaning compute power is unused, also results in a penalty. The objective of the environment is to learn to balance these conflicting problems (**?**).

In the environment the representation of a state is defined by the variables W and N, which represent the workload and number of active CPUs respectively for a particular PoP. It is assumed that every car entails an equal amount of work. Time is measured in five minute intervals. During these intervals it is assumed that the work is distributed over all N active CPUs by a factor of $\frac{1}{N}$ with small random deviations. If the assigned work exceeds the work that can be done, a backlog is created that impairs performance (**?**). Specifically the reward function has the property that it increases as the CPU loads increase, but decreases as the backlogs increase.If there is no backlog the load on each CPU is approximately $\frac{W}{N}$.

4.2.3 Experimental setup

To get the most accurate comparison with (**?**), we will use 80% of the Torino data as training examples, and 20% to run our our experiments on. To measure performance of our A2C approach, we look at the reward and loss at each episode on the test set. To compare the performance of A2C to that of the LSTM and Q-learning in (**?**), we measure on the test set:

- the average reward,
- the average number of CPUs,
- reward per time interval,
- the maximum CPU load per time interval,
- the number of CPUs per time interval

The results that we compare our A2C method on are provided by the authors of (**?**).

The Adam Optimizer (**?**) is used for training with a learning rate of 0.0001. The discount factor is set to $\gamma = 0.5$. The entropy loss factor is set to 0.01. We use Monte

Carlo methods to train our network, because we do not want our model to suffer from potential bias. Our environment is possibly non-Markovian, since the agent does not have knowledge about the entire V2N or of all incoming traffic, while it is dependent on them. To elaborate, two states can look identical to our agent, while two different actions are the optimal choice, meaning at least some information about the environment is hidden. Considering that we use the TD-error to estimate the advantage function, using a one step update would render less accurate values.

4.3 Data-driven ML-based Scaling for V2N Services: Centralized control

To model the auto-scaling problem in a different way, a new environment has been developed by the researchers of (**?**). The previous environment represented a single PoP, which is only a subsection of the entire network, and considered the bulk number of cars entering the area covered by the PoP during the 5-minute time interval. When increasing the scope of the environment to deal with the entire network, we are now dealing with a multi-dimensional problem space. Not only do vertical scaling decisions have to be made for each PoP, but each incoming car is considered separately, thus it has to be assigned to (served by) the most appropriate PoP based on its location. Consequently the problem that is to be solved is no longer a simple scaling problem, but an assignment and scaling problem that can be formulated as an integer program that needs to run every time a car arrives. We also consider a global controller for the entire set of available PoPs in Torino that is taking the corresponding decisions.

We are using the first version of the environment to get preliminary results. These results provide us with insight in whether or not A2C is a viable method for addressing the problem at hand while it might also give insight in possible points for improvement. To make this environment more general and to encourage further research with it, we develop a version that is consistent with the OpenAIGym (**?**) format.

In this section we first discuss what this new environment entails, second we discuss what steps are taken to apply A2C on this environment, third we describe what steps are necessary to make the environment comply to the OpenAIGym format, and fourth we describe our experimental set-up to acquire the preliminary results.

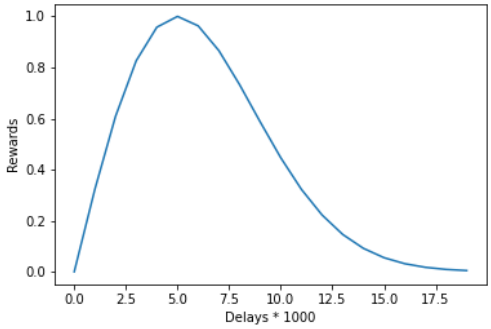

Figure 4.1: Reward function

4.3.1 Fifty-stations environment

In this environment there are fifty PoPs that all need to be managed by the agent. In the context of this environment we will refer to PoPs as stations. Each incoming car needs to be placed at a station, and a vertical scaling decision needs possibly to be made at that station, in order to facilitate the additional computing requirements imposed by the incoming car.

Along with developing this environment, an accompanying data trace was developed. This trace contains three features for each datapoint, namely 'longitude', 'latitude', and 'arrival time'. Each longitude and latitude pair corresponds to a station. Episodes are defined as entire passes over the dataset.

At each time-step, a new car arrives in the network. This incoming car can be placed at any station in proximity, meaning that this does not have to be the station where it arrived to. For instance this could be the case when this station is already very busy. Placing a car at a different station than the station closest to it when it enters the network results in a small delay penalty. The environment also provides for vertical scaling of any of the stations. This can be done between a minimum and maximum CPU parameter.

This environment utilises a delay-based reward, based on the time the . Both the action dimensions can result in a delay. Whenever a car is placed or a CPU is set at a station there is a certain delay. Optimal delay is assumed to be around 0.005 seconds. The assumption is that a lower delay could point towards an overprovisioned or inactive CPU, while a higher load could point towards an overloaded queue of cars. The reward function should be maximised and has a range of 0 to 1, its behaviour is shown in 4.1.

4.3.2 Applying A2C to novel environment

To apply A2C on the new environment, we have to build an intermediate layer that manages state and action representation as well as interaction with the A2C networks.

To represent the state, we use a matrix with two columns. Each row represents one station. The first column represents the amount of active CPUs, while the second column represents the amount of cars at that station. We use the amount of cars instead of the delay, because the delay calculation can return infinite. Since the amount of work per car is assumed to be equal, the amount of cars at a queue is also a meaningful feature.

To represent the actions in a way that is easily interpreted by a neural network, we use a tuple representation. One member represents the station at which the incoming car needs to placed, while the second member represents the new amount of CPUs at that station. The action to place a car at a station implies that the queue at that station is increased with the delay of an additional car. To account for a two-dimensional action, the A2C actor has to be modified so it can output two action-distributions.

4.3.3 OpenAIGym format

We want an environment, that any RL agent learning can use and interact with. Consequently, a standard API in the OpenAI gym format (**?**) is implemented. This means that there should be an *init, step, reset, render*, and *close* function.

- The *init* function instantiates an instance of an environment, here the action and observation space are defined. These are OpenAIGym attributes that define the format and range of the actions and states the agent and environment can take on respectively.
- The *step* function takes an action as input and returns the new state, reward, done flag, and optional information. This function takes an action and transforms an action, calculates the corresponding reward, checks if the episode is finished and then returns.
- The *reset* function puts the environment back in its initial state and should be called at the start of each episode. It returns a state that is at the start of an episode.
- The *render* function displays the current state in a pop-up window.
- The *close* function should close the visualization window that was created when calling the *render* function.

The *render* and *close* functions are left for later development as they are not required for preliminary results.

4.3.4 Experimental setup

To acquire preliminary results on the new station we use a part of the tracefile to speed-up training. We use 1000 measurement points. We use %80 for training and %20 for testing purposes. We utilise the same settings as in experiments on the Torino environment to be able to analyse difference of environment on learning success. The Adam Optimizer (**?**) is used for training with a learning rate of 0.001. The discount factor is set to $\gamma = 0.5$. The entropy loss factor is set to 0.01. We use Monte Carlo methods to train our network.

5

Results

5.1 A2C implementation and benchmarking

Figure 5.1 depicts the results of the A2C implementation applied to the cartpole environment from OpenAIGym. On the left-side, the episode length is plotted for each episode, on the right side rewards per episode are plotted for each episode. In blue is the actual value, the orange represents the moving average.

Our A2C implementation solved the cart-pole balancing problem in 502 trials. However, convergence is unstable and some runs took several hundreds of episodes longer. In the presented results the discount factor γ is set to 0.998 and the entropy factor is set to 0.01. The learning rate is set to $1e-03$ and the hidden layer size is set to 16. It can be observed in 5.1 that both episode length and reward have a sharp increase around 200 episodes, after which it takes a little more than 200 episodes to be solved.

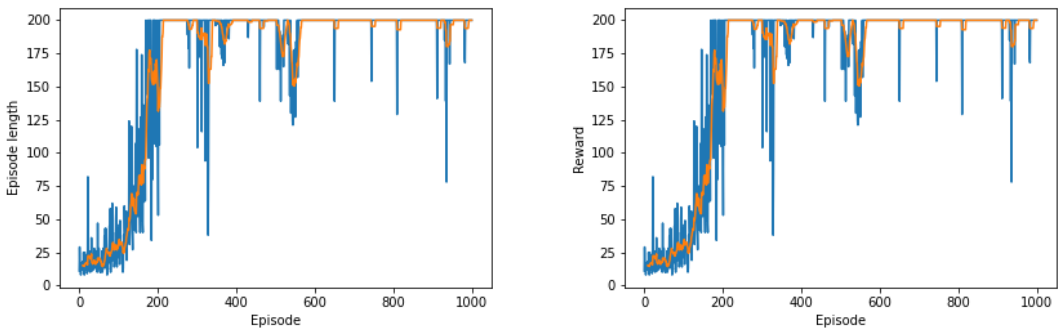

Figure 5.1: Cart-pole balancing task

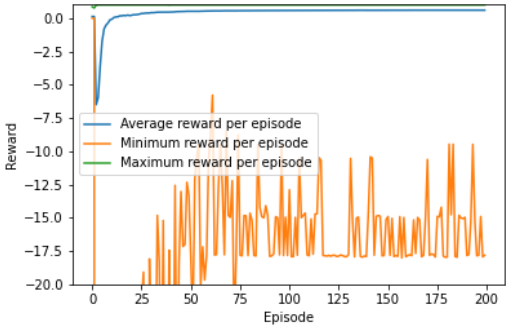

Figure 5.2: Scaling C-V2N services

5.2 Data-driven ML-based Scaling for V2N Services

Figure 5.2 presents the rewards per episode on the test set for A2C. As can be observed in the figure, the maximum reward increases from 0.85 to around 1.0 at around episode 10, meaning that the agent learns to take more optimal actions relatively fast. The minimum reward has a high variance and is sometimes extremely low, more often so in the earlier episodes of the test set. This shows that wrong actions can have a highly volatile effect. The average reward shows an increase from around -7.00 to 0.59. This shows that the agent takes better decisions on average the longer it learns. Most of the increase in rewards happen in the first 50 episodes on the test set.

Figures 5.3 to 5.6 compare the A2C method to the LSTM and Q-learning approach employed in (**?**). To acquire these results, we have reproduced the Q-learning results for the specific environment that our A2C model was trained on. However, currently the LSTM results are not reproducible. Consequently, the results from the LSTM are taken from (**?**), which are based on an older version of the environment.

In 5.3 it can be observed that the Q-learning approach has a reward of around 0.36, the LSTM an average reward of 0.54, and the A2C an average reward of 0.59. The A2C approach uses the lowest average number of CPUs (10), while the Q-learning approach uses 23, and LSTM 13.

Figures 5.4 to 5.6 in particular, illustrate how the three algorithms perform over a period of two days (or equivalently, 576 5-minute intervals) of the test trace.

In figure 5.4 it can be observed that A2C consistently has highest and most variable CPU load, while following a very similar pattern to the LSTM. The Q-learning has a significantly lower CPU load overall but still follows a similar trend compared to the other methods.

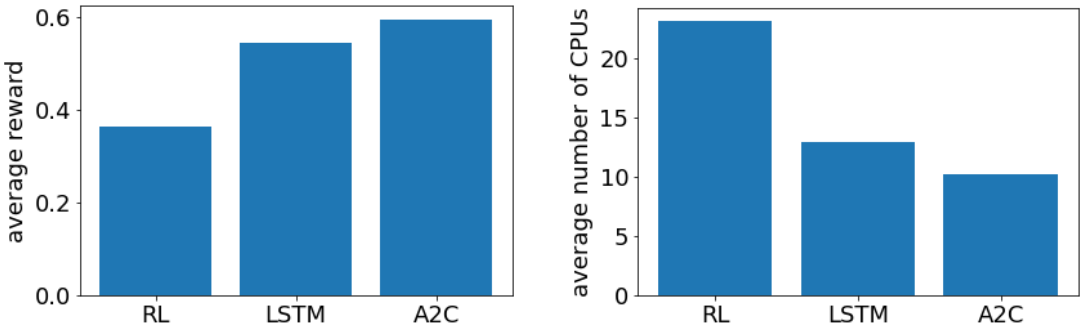

Figure 5.3: Average performance metrics.

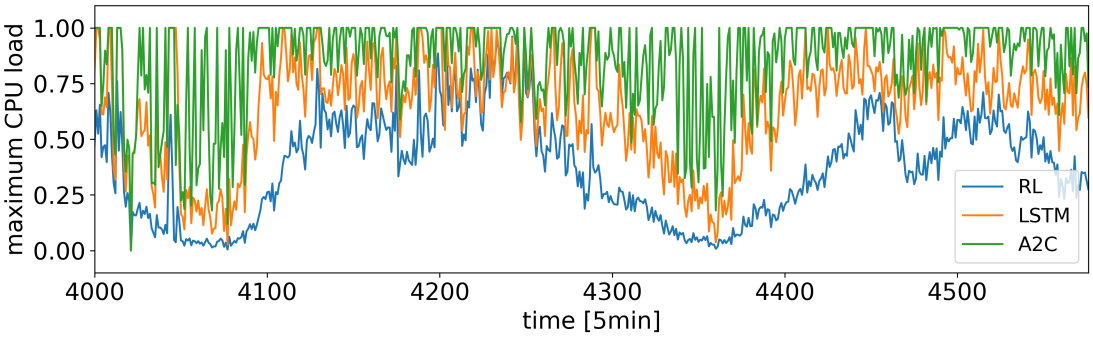

Figure 5.4: Maximum CPU load over time

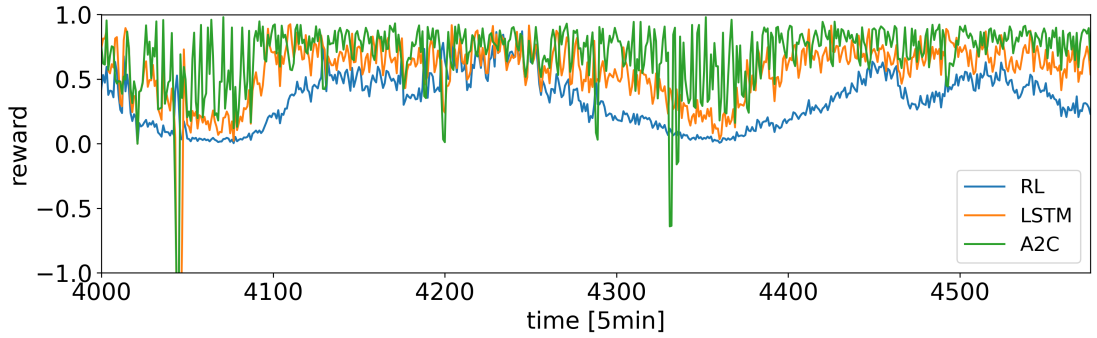

Figure 5.5: Rewards over time

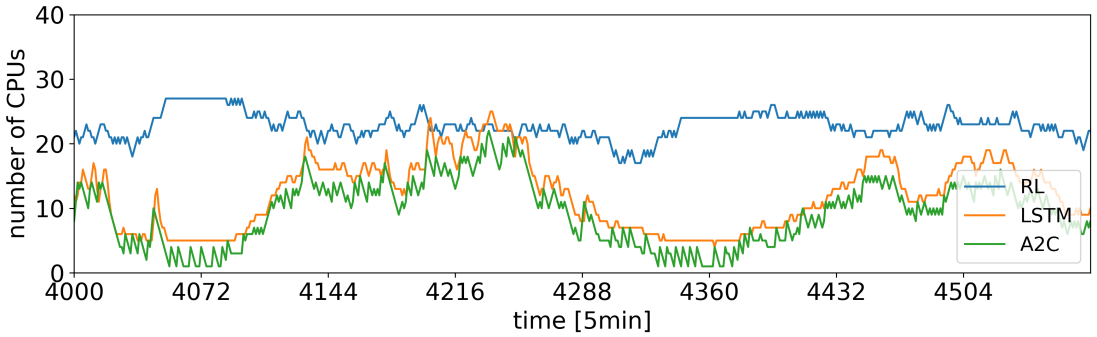

Figure 5.6: Number of CPUs over time

The rewards for the A2C implementation and the LSTM implementation have very similar rewards on the test set as can be observed in 5.5. The Q-learning implementation has a more irregular but also lower reward overall than the other two approaches at at almost at every time-step.

As shown in figure 5.6, our A2C approach behaves very similar to the LSTM approach. They seem to follow exactly the same pattern, only the A2C makes more scaling decisions. The Q-learning approach has a higher amount of active CPUs at almost every timestep and seems to stay consistently around 23 CPUs.

Overall, the RL controller is the most conservative as it does not decrease the number of processors, but hardly ever overloads any CPU. The A2C controller is the most aggressive in this set of experiments as it activates the smallest number of CPUs on average and while frequently a CPU gets overloaded, as this is mostly of short duration it does not have an

impact on the reward. The LSTM controller has a behaviour between these two extremes.

5.3 Data-driven ML-based Scaling for V2N Services: Centralized control

Figure 5.7 presents results of the A2C implementation applied to the extended version of the scaling problem at hand. On the left-side, rewards per episode are plotted for each episode while on the right-side the delays per car on the test set are shown. It can be observed in 5.7 that the rewards on the test set go towards zero the longer our model runs. The maximum as well as the average reward become 0 around episode 150 onward. This happens consistently every time we train the A2C on this environment. This can either mean that the delay goes to 0.0 or to infinity. The average delay of this run was 0.0051, which is very close to the delay target 0.005. However, acquiring these results also sometimes results in an average delay of 'infinite' value. Which can happen if one of the delays got an infinite penalty. The particular plot of delays that is shown here is of a more optimal test where no delay went to infinity.

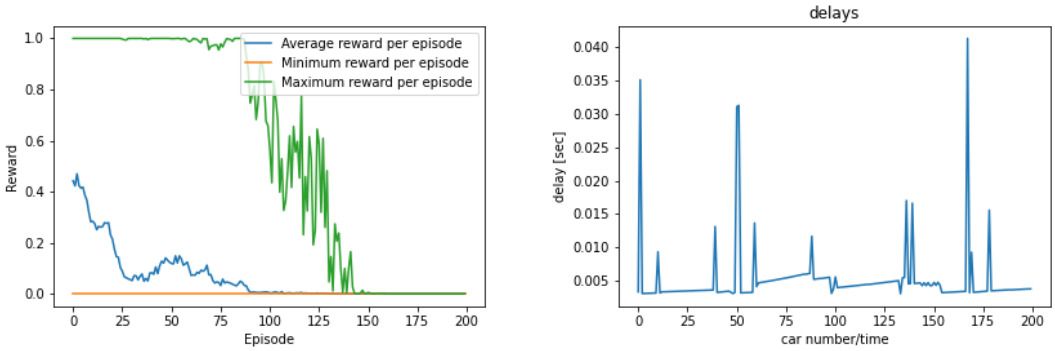

Figure 5.7: Car placement and scaling C-V2N services.

6

Discussion

6.1 Data-driven AI-based Resource Scaling

The results show that A2C methods have the potential to be successful in the Torino environment developed in (**?**). A2C showed a similar behaviour to the LSTM, but outperformed it slightly in terms of average reward. An average reward as high as the LSTM was already reached after 50 training episodes, while the maximum average reward was only reached much later. This shows that A2C can be a very successful ML approach in network slice elasticity and it exposes a dilemma where accuracy and training cost prior to model deployment should be carefully weighed.

In our results it appears that the A2C keeps the number of active CPUs low, while often makes the decision to scale up or down. The relatively low amount of active CPUs in the PoP results in a higher maximum CPU load. This shows that the A2C has a tendency to minimise idleness, but that it is more prone to overloading CPUs as well. The scaling decisions that go up and down very fast could create much overhead in a real-world scenario, which might have to be taken into account in an environment to get better results if this model was to be deployed.

In conclusion, A2C can be successfully applied to the problem of network slice elasticity by ways of vertical scaling in a V2N network environment where an agent manages one PoP. However, there are still some challenges that need to be overcome to be sure that it can also work in a real life setting. Further research should focus on making a learning environment as realistic as possible and assessment of cost and benefits of using these data-driven approaches. Furthermore, future research should look into the balancing of accuracy versus training costs.

6.2 Data-driven ML-based Scaling for V2N Services: Centralized control

The preliminary results on the first version of the new environment show that consequently the reward goes to zero consistently on the test set. Since this could mean that the delay goes to either zero or infinity, it can mean that the agent learns to behave optimally or sub-optimally. A sub-optimal action in an episode can lead to infinite delay, and then a reward of zero. While a perfect action leading to zero delay can also lead to a reward of zero. The optimal delay is set to a delay target, which was 0.0051 in this case. This could complicate learning since it is such a small interval to succeed in. Specially if the effect on the delay by adding or removing a CPU is of the same order of magnitude as the desired delay. When we look at the delay on the cars in the test set, many of them are almost exactly at the delay target. This shows that the A2C is learning to place cars in a way that is optimal for this environment. The fact that the delays are optimal on average, but the reward is 0.0 on average shows that the rewards per episode are not a sufficient metric for this representation of the problem. Furthermore it points to the need to further optimise the reward function, since it can be hard for a neural network to generalize a function where the optimum is on a very sharp peek. This is because any slight misstep can result in a low reward. Another worthwhile addition to the reward function would be to incorporate a penalty that reflects the cost of activating a CPU, as to more accurately reflect the actual situation.

However, the delays of cars in the test set was not always stable and this could be improved on in further research. A good starting point for this would be to use the full dataset. Other approaches could be trying to stabilise A2C further, attempting a ML approach that has less variance, or trying to tune the reward function so a different approach is learned by the A2C.

Concluding, A2C is a viable approach for further research into successful ML methods for auto-scaling of network slices. The first results for the new environment show that an A2C is capable of learning to successfully place cars and scale PoPs simultaneously. Utilizing a ML method for scaling as well as placement of cars should further investigated because as ML methods are able to potentially abstract relationships in extremely large datasets which are too complicated for humans to analyse.

Appendix

A link to all the code that was used and created for this thesisis available at: https://github.com/WillemijnBeks/master_thesis

References

[] Ibrahim Afolabi, Tarik Taleb, Konstantinos Samdanis, Adlen Ksentini, and Hannu Flinck. Network slicing and softwarization: A survey on principles, enabling technologies, and solutions. *IEEE Communications Surveys & Tutorials*, 20(3):2429–2453, 2018.

[] Zafarali Ahmed, Nicolas Le Roux, Mohammad Norouzi, and Dale Schuurmans. Understanding the impact of entropy on policy optimization. In *International Conference on Machine Learning*, pages 151–160. PMLR, 2019.

[] NGMN Alliance. 5g white paper. *Next generation mobile networks, white paper*, 1, 2015.

[] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[] NM Mosharaf Kabir Chowdhury and Raouf Boutaba. A survey of network virtualization. *Computer Networks*, 54(5):862–876, 2010.

[] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[] Danny de Vleeschauwer, Jorge Baranda, Josep Mangues-Bafalluy, Carla Fabiana Chiasserini, Marco Malinverno, Corrado Puligheddu, Lina Magoula, Jorge Martín-Pérez, Sokratis Barmpounakis, Koteswararao Kondepu, et al. 5growth data-driven ai-based scaling. In *2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, pages 383–388. IEEE, 2021.

REFERENCES

[] Netherlands Authority for Consumers and Markets (ACM). 5g and the netherlands authority for consumers and markets, 2018. URL `https://www.acm.nl/sites/default/files/documents/2018-12/5g-and-acm.pdf`.

[] David M Gutierrez-Estevez, Marco Gramaglia, Antonio De Domenico, Nicola Di Pietro, Sina Khatibi, Kunjan Shah, Dimitris Tsolkas, Paul Arnold, and Pablo Serrano. The path towards resource elasticity for 5g network architecture. In *2018 IEEE wireless communications and networking conference workshops (WCNCW)*, pages 214–219. IEEE, 2018.

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

[] Yuxiu Hua, Rongpeng Li, Zhifeng Zhao, Xianfu Chen, and Honggang Zhang. Gan-powered deep distributional reinforcement learning for resource management in network slicing. *IEEE Journal on Selected Areas in Communications*, 38(2):334–349, 2019.

[] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[] Rongpeng Li, Zhifeng Zhao, Qi Sun, I Chih-Lin, Chenyang Yang, Xianfu Chen, Minjian Zhao, and Honggang Zhang. Deep reinforcement learning for resource management in network slicing. *IEEE Access*, 6:74429–74441, 2018.

[] Rongpeng Li, Chujie Wang, Zhifeng Zhao, Rongbin Guo, and Honggang Zhang. The lstm-based advantage actor-critic learning for resource management in network slicing with user mobility. *IEEE Communications Letters*, 24(9):2005–2009, 2020.

[] OpenAIGym. Cartpole-v0. `https://gym.openai.com/envs/CartPole-v0/`, 2021.

[] Chrysa Papagianni, Josep Mangues-Bafalluy, Pedro Bermudez, Sokratis Barmpounakis, Danny De Vleeschauwer, Juan Brenes, Engin Zeydan, Claudio Casetti, Carlos Guimarães, Pablo Murillo, et al. 5growth: Ai-driven 5g for automation in vertical industries. In *2020 European Conference on Networks and Communications (EuCNC)*, pages 17–22. IEEE, 2020.

[] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga,

Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[] Sabidur Rahman, Tanjila Ahmed, Minh Huynh, Massimo Tornatore, and Biswanath Mukherjee. Auto-scaling vnfs using machine learning to improve qos and reduce cost. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2018.

[] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

[] Tejas Subramanya and Roberto Riggio. Machine learning-driven scaling and placement of virtual network functions at the network edges. In *2019 IEEE Conference on Network Softwarization (NetSoft)*, pages 414–422. IEEE, 2019.

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

[] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4): 279–292, 1992.

[] Wenfeng Xia, Yonggang Wen, Chuan Heng Foh, Dusit Niyato, and Haiyong Xie. A survey on software-defined networking. *IEEE Communications Surveys & Tutorials*, 17(1):27–51, 2014.

[] Mu Yan, Gang Feng, Jianhong Zhou, Yao Sun, and Ying-Chang Liang. Intelligent resource scheduling for 5g radio access network slicing. *IEEE Transactions on Vehicular Technology*, 68(8):7691–7703, 2019.

[] Zheng Zhao, Weihai Chen, Xingming Wu, Peter CY Chen, and Jingmeng Liu. Lstm network: a deep learning approach for short-term traffic forecast. *IET Intelligent Transport Systems*, 11(2):68–75, 2017.