POLITECNICO DI TORINO

Department of Electronics and Telecommunications
Master degree course in Communications and Computer Networks Engineering

Master Degree Thesis

AI-based algorithms and experimental evaluation for beyond 5G

Supervisor
prof. Carla Fabiana CHIASSERINI
prof. Claudio Ettore CASETTI

Candidate
Federico MUNGARI

ACADEMIC YEAR 2019-2020

Abstract

The fifth-generation technology standard for cellular networks (5G) aims to provide high throughput, reduced latency, massive connection and a shift from service-orientation to user-orientation in requirements and innovations. To achieve the target goals, efficient resource allocation and management are required. In this regard, the concept of Network Function Virtualization (NFV) has been introduced, as well as the one of Software Defined Network (SDN). NFV is a state-of-the-art approach which replaces whole classes of network's nodes functions, which were generally deployed on dedicated hardware, with pure software implementations. The virtualization of radio access networks (vRAN) comes under the NFV paradigm. The goal of vRANs is to centralize the virtualized radio access point (vRAP) processing stack, and to efficiently map the users' requirements into radio and computational resources allocation. This progress is required by mobile operators in order to support the rising traffic demand and ensure the varied quality-of-service (QoS) requested at affordable cost. But currently, due to the high complexity of the dependencies between computing and radio resources, there still are not sufficiently efficient solutions to satisfy the telecommunications industry.

In this work it is investigated and presented the design of a dynamic resource controller for vRANs based on machine learning techniques. Such a design allows to continuously adapt the resource allocation to the actual demand across vRAPs.

In our configuration, the first acting entity is an encoder which accomplishes the task of translating the high-dimensional context information into a simpler representation. The context information is summed up in a vector per RAP containing samples of the traffic load to be managed and of the average and variation of the signal-to-noise ratio experienced by the users. It has been chosen to implement an independent deep autoencoder with nonlinear activation functions and sparsity constraint per context feature, in order to get the temporal correlation between its consecutive samples.

The encoder is then followed by the proper resource manager, which has the duty of understanding the relationship between radio solutions and the corresponding computational load. The resource manager deployment is based on deep reinforcement learning techniques and it aims at the maximization of a reward signal which is proportional to the fulfilment of QoS requirements, traffic reliability and computational resources savings.

The resource controller is composed by a CPU and a Radio policy. The first one is responsible for the distribution of the computational resources among all the RAPs. The last one is instead responsible for the choice of a bound for the modulation and coding schemes (MCS) to be adopted by each RAP, which strictly depends on the available computing capability.

For what concerns the CPU policy, it has been designed a deep deterministic policy gradient algorithm based on an actor-critic model-free solution. While the actor neural networks is tasked with the choice of the CPU allocations, the critic provides the estimates for the received rewards. Such a model efficiently manages to cope with real valued actions and high-dimensional state spaces.

The Radio policy instead relies on an iterative procedure in which a deep binary classifier judges in descending order different values for the upper bound to the eligible MCSs. The latter choice is fundamental in order to avoid incurring heavy deterioration in terms of system performance caused by CPU outages.

The proposed design of a dynamic resource controller is a very promising one that potentially meets the stringent needs of telephone operators for a virtual access network that is able to meet the demands of the heterogeneous spectrum of users.

Contents

List of Figures

List of Acronyms

AI	Artificial Intelligence
BBU	Baseband Unit
BS	Base Station
BSR	Buffer Status Report
CAPEX	CAPital EXpenditure
CB	Contextual-Bandit
CDMA	Code Division Multiple Access
CN	Core Network
CQI	Channel Quality Indicator
cRAN	Centralized Radio Access Network
DDPG	Deep Deterministic Policy Gradient
DP	Dynamic Programming
DQN	Deep Q Network
e-UTRAN	evolved UMTS Terrestrial Radio Access Network
EPC	Evolved Packet Core
FDD	Frequency Division Duplex
FFT	Fast Fourier Transform
GSM	Global System for Mobile Communications
ITU	International Telecommunication Union
LTE	Long Term Evolution
MCS	Modulation-and-Coding Scheme

MDP	Markov Decision Process
MEC	Multi-access Edge Computing
ML	Machine Learning
MSE	Mean Squared Error
NFV	Network Function Virtualization
NN	Neural Network
OFDM	Orthogonal Frequency-Division Multiplexing
OPEX	OPerating Expenditure
P-GW	Packet Data Network Gateway
PUSCH	Physical Uplink Shared Channel
QoS	Quality of Service
RAP	Radio Access Point
RAN	Radio Access Network
RB	Resource Block
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RRC	Radio Resource Control
RRH	Remote Radio Head
RRU	Remote Radio Unit
SAE	Sparse Autoencoder
SDN	Software Defined Network
S-GW	Serving Gateway
TDMA	Frequency Division Multiple Access
TDD	Time Division Duplex
UE	User Equipment
USRP	Universal Software Radio Peripheral
vRAN	Virtual Radio Access Network
vRAP	Virtual Radio Access Point

Chapter 1

Introduction

Global mobile data traffic is exponentially growing. The Cisco visual networking index [1] reports an estimate of future global IP traffic growth and trends. Global IP data traffic is expected to grow 11-fold from 2012 (437 exabytes [1]) to 2022, and 3-fold from 2017 (122 exabytes per month) to 2022. This forces the mobile networks to handle more and more amount of traffic. In order to support and actuate the always increasing client necessities in the field of mobile phone communication, both in terms of performance requirements and number of subscribers, there must necessarily correspond a network evolution. Network providers need to constantly update and deploy new advancements for the mobile networks. That is why, after about every of the last five decades, a new generation of mobile network with always better features has been set up.

Here it is presented a short overview of the evolution of mobile generations, in order to appreciate the important progress and growth that the communication world has experienced and is experiencing.

1G In the early 1980s the 1st generation system came into existence. Commonly it is referred to as 1G. It was an analogue-based communication technology which offered only analogue voice services relying on a circuit switching network. The supported maximum data rate was tremendously: 2.4kbps. Moreover, the affordable number of calls at the same time was very bounded too, also because of the inefficiency in radio resource exploitation.

2G The 2nd Generation (2G) mobile networks is the second stage of the wireless systems. The main differences with respect to the first generation are that 2G is completely digital and it offers also data services, like Short Message Services (SMS) and Multimedia Messaging Service (MMS). Moreover, it provides enhanced calling features like caller ID. 2G systems offered a larger data bandwidth up to 64kbps. The most popular 2G system is the Global System for Mobile Communications (GSM), whose digital access techniques it the Time Division Multiple Access (TDMA) with three time slots in each 30KHz channel.

[1]An Exabyte is one billion Gigabytes.

2.5G Next to 2G, 2.5G system relies on both packet switched and circuit switched domains. The provided data rate is improved with respect to 2G up to 144 kbps. The code division multiple access (CDMA) system was adopted in America while in Europe an upgrade from GSM to GPRS and EDGE systems was realized.

3G Differently from the previous two generations, the 3rd generation is an international standard released from ITU (International Telecommunication Union) which was named Universal Mobile Telecommunications System (UMTS). 3G brought about a great progress and enhancement in the telecommunication area: not only it improved the maximum data rate up to 2Mbps and enlarged the available capacity, but also the range of data services and application was widened. Indeed, the benefits of higher data rates and greater bandwidth mean that 3G mobile phones can offer subscribers services such as mobile Internet access and multimedia applications, and so a 3G handset provides many new features such as TV streaming, multimedia, videoconferencing, Web browsing, e-mail, paging and navigational maps [2]. That is why 3G can be considered a system that has mainly a focus of data, rather than voice. Such an improvement was possible thanks to new features introduced in the 3rd technology. First of all, the most adopted radio interface is called W-CDMA (Wideband Code Division Multiple Access). The exploited bandwidth is of 5 Mhz. And both frequency (FDD) and time division duplexing (TDD) could be accepted – in Europe the FDD option was exclusively chosen. This multiple access technology allowed to efficiently exploit the radio resources, since different BS can be superimposed in both time and frequency, and so exploit the same radio resources. The achievable performance of 3G systems is: data bandwidth of up to 144 kbps for high speed moving users, up to 384 kbps for pedestrians and up to 2 Mbps for indoor or stationary users.

4G The 4th Generation (4G) mobile networks meant a transition from low to high data rate. A stationary user can indeed achieve up to 1Gbps data rate, while a moving vehicle is limited to 100Mbps. The most popular 4G system was presented for the first time in 2009: it is the Long Term Evolution (LTE) system, which adopts OFDM (Orthogonal Frequency Division Multiplexing) multiplexing technique. This technique consists of multiple closely spaced orthogonal subcarrier signals with overlapping spectra whose demodulation is based on fast Fourier transform algorithms. Not only LTE improved the connection speed, but also the network capacity and its latency (few milliseconds), by means of an IP-based and simplified network architecture. The 4G enhancement promises to bring the wireless experience to an entirely new level with impressive user applications, such as sophisticated graphical user interfaces, high-end gaming, high definition video and high-performance imaging [2].

The fifth generation of mobile networks will not be a simple improvement of the previous generations; it instead will be revolutionary in all respects: data rates [2], latency[3], area traffic capacity, connection density, network reliability, and energy efficiency. All these very demanding targets are clearly stated in [3] and showed in figure 1.1.

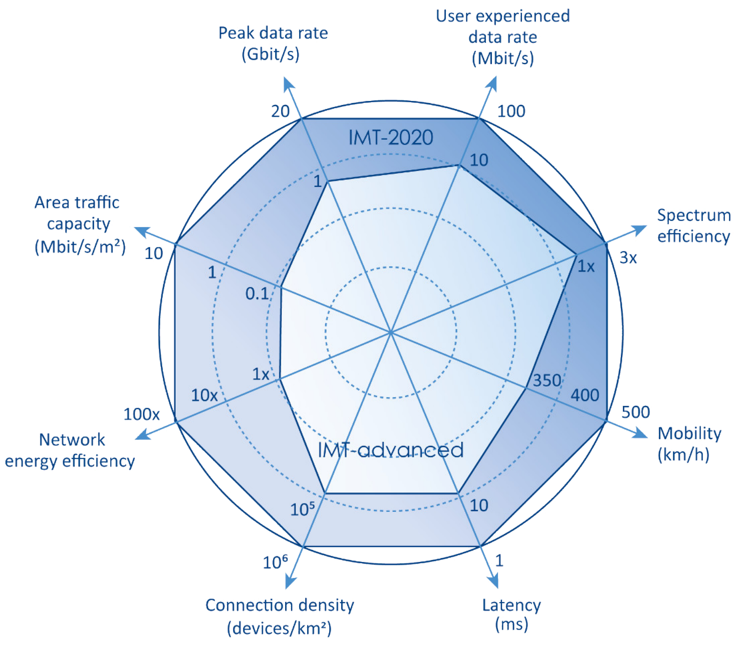


Figure 1.1: 5G key features

These features are aimed at achieving not only ultra-high-speed-data, but also super high-capacity for the Internet of Things, augmented virtual reality, the tactile internet, and so on. In the future society, everyone and, remarkable, everything will be interconnected. With the advent of 5G, we migrate from a single-discipline system to a multi-discipline one.

International Telecommunication Union (ITU) has summarized a diverse variety of usage scenarios/use cases in three broad categories to be supported from 5G.

eMBB - Enhanced Mobile Broadband it is a data-driven use case for serving mobile users requiring high data rates. This is the 4G use case, but with improved performance and an increasingly seamless user experience.

mMTC - Massive Machine Type Communications it deals particularly with connectivity and networking amongst massive numbers (billions) of machines and sensors. It is considered a key progression from the Internet of Things IoT to the Internet of Everything. This type of devices does not require high bit rate, but there are

[2]5G must be able to provide a user experience data rate 10 times higher and a peak data rate 20 times higher than those which are currently available.

[3]5G must be able to provide a user experience latency reduced by an order of magnitude with respect to 4G.

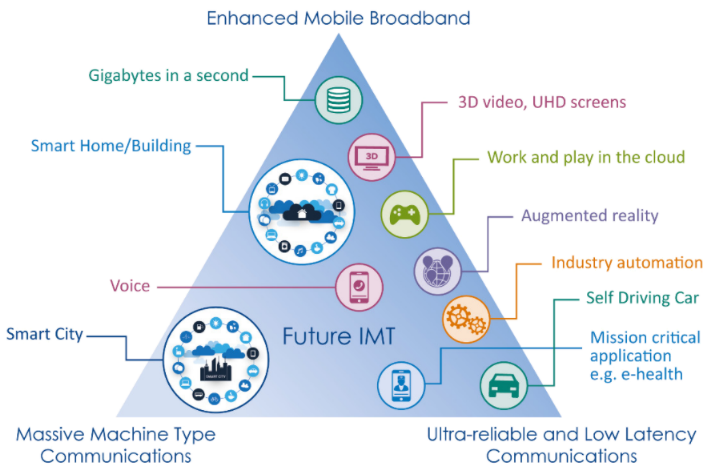

Enhanced Mobile Broadband
Gigabytes in a second
3D video, UHD screens
Smart Home/Building
Work and play in the cloud
Augmented reality
Industry automation
Voice
Self Driving Car
Smart City
Future IMT
Mission critical application e.g. e-health
Massive Machine Type Communications
Ultra-reliable and Low Latency Communications

Figure 1.2: 5G usage scenarios

restrictions in terms of network coverage underground and extended battery-life;

URLLC - Ultra-reliable and Low Latency Communications this use case has stringent requirements for capabilities such as latency, reliability and availability also. Some practical examples that fall into this case are tactile internet applications, transportation safety, smart grid and remote medical surgery.

The very demanding targets of 5G are fulfilled exploiting large bandwidths at extremely high frequency: the so-called mm-wave bands. This allows to introduce some other technologies and advancements as the 3-D MIMO, which needs antenna arrays, and network densification. Also, scalability and flexibility are needed so as to support all the different use cases: consequently, the 5G network must be ductile and adaptable, namely service-based, differently from the static point-to-point architecture currently in use by 4G.

In this regard, the separation of Control Plane Functionalities and User Plane ones is required. And this decoupling can be realized by means of network virtualization techniques:

Software Defined Network (SDN) it allows to abstract, with the provisioning of programmability over the entire network, a centralized control plane that manages the user plane functions. Moreover, SDN benefits from standard protocols for managing and re-programming the network on the fly, optimizing the performance;

Network Function Virtualization (NFV) it is the process of relocating, decoupling the abstraction of functionalities from the corresponding hardware, namely to run the network functions (the so called Virtual Network Functions – VNFs) such as routers and firewalls on a general-purpose server.

SDN and NFV are two strictly related technologies, even if they could be used individually. Nevertheless, the complementary deployment of both the paradigms can introduce considerable benefits.

The virtualization of radio access networks (vRAN) comes under the NFV paradigm. The goal of vRANs is to centralize the virtualized radio access point (vRAP) processing stack, and to efficiently map the users' requirements into radio and computational resources allocation. This progress is required by mobile operators in order to support the rising traffic demand and ensure the varied quality-of-service (QoS) requested at affordable cost. But currently, due to the high complexity of the dependencies between computing and radio resources, there still are not sufficiently efficient solutions to satisfy the telecommunications industry.

Chapter 2

Radio Access Network of 5G systems

Virtualized Radio Access Network (vRAN) architecture is the solution that grant to the network self-optimization, self-configuration and self-adaptation in software control capabilities, through SDN and NFV paradigms. The conventional RAN architecture widely deployed today can hardly afford the additional capacity, cost savings, service agility, and scalability that service providers need to meet future demand.
The RAN, and its functionalities also, can be split in two main units:

- **Remote Radio Unit (RRU)**: this module performs all the RF functionalities like transmit and receive functions, filtering, and amplification. It also contains analogue-to-digital or digital-to-analogue converters.
- **BaseBand Unit (BBU)**: it provides the physical interface between the base station and the core network, managing the whole Base Station (BS) system, including operating, maintenance and signalling processing. The main functions are coding, modulation, Fast Fourier Transform (FFT) and radio resource control (RRC).

2.1 LTE RAN limits

Three main components can be identified in an LTE cellular network architecture: the cellular Radio Access Network (RAN), the cellular Core Network (CN) and, finally, the Internet. The RAN is composed by the Base Stations (BSs), which are also called eNodeB in LTE. The CN instead can be summarized by two main blocks, namely the Serving-Gateway (S-GW) and the Packet-Gateway (P-GW). What was stated is shown in figure 2.1.
In the first LTE architecture, the BBU and the RRU – also called Remote Radio Head (RHH) – are physically separated and spaced by several km. This architecture - which is shown in figure 2.2 - was introduced when 3G networks were being deployed and right now the majority of base stations still use it. RRUs are statically assigned to BBUs. In traditional architecture, instead, which was deployed for 1G and 2G cellular networks, both RRU and BBU were integrated and placed with the BS.

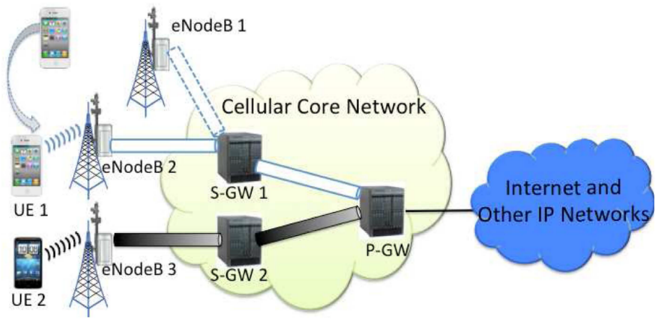


Figure 2.1: LTE architecture

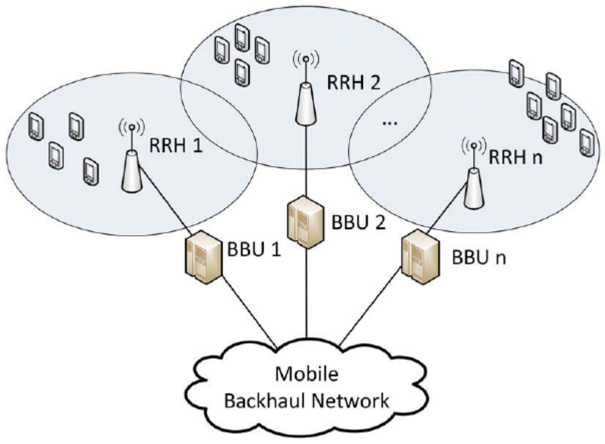


Figure 2.2: RAN with RRH

The benefit of the LTE RAN architecture with respect to the traditional one was that the BBU could be placed in a more convenient and accessible place, leading to cost savings in operating and rental. Moreover, one single BBU can serve a group of RRUs.

According to the architecture of figure 2.1, the user equipment (UE) establish a connection with the eNodeB, which forward traffic to the S-GW. This latter must provide to the user seamless connection, regardless its mobility. P-GW, instead, enforces QoS policies and has monitoring tasks, e.g. acting like a firewall against the undesired connections. It follows that BS, S-GW and P-GW, not only accomplish data plane functionalities, but also control plane ones. At first, they have indeed to manage the connections, which involve the setup, tear-down and reconfiguration phases. Then they are in charge of support user mobility (registration, paging and handoff). Moreover, the CN is also responsible for some QoS aware routing decisions.

Under the light of the above, two main problems arise.

Scalability challenges all traffic is forced to be managed by the P-GW. Not only this makes the P-GW hardware tremendously expansive, but also hosting popular contents becomes unreliable;

Vendor specific device configuration LTE cellular systems are composed by closed hardware components with vendor-specific interfaces. That means that the operators do not have direct control on their networks. Consequently, the possibility of improving the current network is tremendously bounded. Operators can not apply new stratagems or technique to boost the network if not buying new expansive hardware. And the idea of dynamically adapt the network for the heterogeneous spectrum of services and for the highly variable traffic patterns remains unthinkable.

In addition, low power consumption, agile traffic management and high reliability are important for today's network architectures.

2.2 Virtualized-RAN

As explained in more detail in [4], the vRAN architecture is characterized by the clustering and virtualization of many BBUs into one single entity: the so called BBU Pool. This latter is installed in a general-purpose high-performance commodity server hardware and consequently shared among several BSs – which will be equivalently called Radio Access Point (RAP). This allows to avoid a strong imbalance between heavily and lowly loaded BBUs. The cloud-computing platform (the BBU Pool), depending on the scenario, can be placed in edge nodes close to the antennas – as in Multi-access Edge Computing (MEC) – or in the CN – centralized RAN (C-RAN). The resulting architecture, which in the following will be referred to as vRAN, is shown in figure 2.3. Obviously, a high bandwidth and low latency transport link is needed between the RRUs and the BBU Pool.

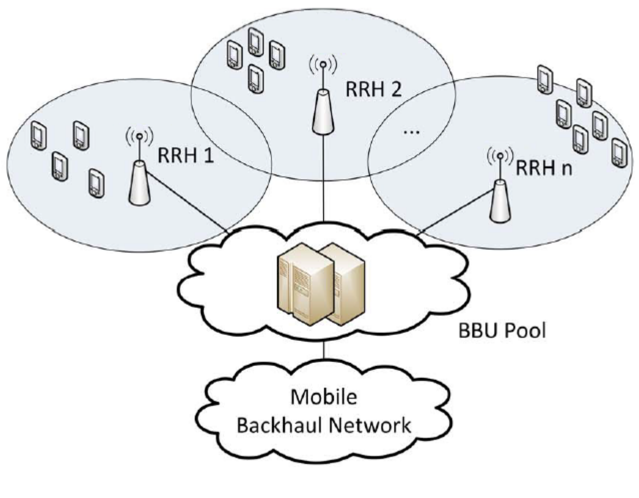


Figure 2.3: Virtualized-RAN architecture

The important benefits that vRAN brings are many, both for macro and small cells.

Adaptability to nonuniform traffic The network load experiences tremendous temporal and spatial traffic fluctuations: the traffic load noticeably varies throughout day, as well as between different areas. But in order to seamlessly work, the network has to be dimensioned over the peak traffic loads, causing a network underutilization at less than 40% maximum load. To name but one example, residential areas experience peak traffic loads in the evening, while commercial and office areas during the morning and the early afternoon. By adopting the vRAN architecture, so by centralizing the BBUs of heterogeneous areas in one single pool, the deriving advantage is the so called multiplexing gain, according to which the computational resources needed by the BBU Pool is smaller than the sum of the individual BBUs capacities that would have been deployed with the distributed RAN. This gain can be especially achieved when small-cell networks are considered. In [5] it has been shown that the peak of aggregated user data traffic in BBU Pool can be a quarter of the sum of peak user data throughputs in BBUs assigned to each of the cells. In this work thesis, the dimensioning of the BBU Pool computational capabilities is not considered.

Scalability vRAN also allows to increase the network flexibility. Indeed, the addition of new RRUs under the same BBU Pool can address the problems of lack of coverage and network capacity poorness. By improving the BBU Pool performance, either by adding more hardware or upgrading the existing one, it is possible to enhance the total network capacity. Obviously, resource problems arise, since BBU Pool computational capacities must be managed and controlled.

Cost savings BSs, which are the main components of wireless access networks, usually represent the largest monetary investment associated with a mobile network [6], both in terms of CAPital EXpenditure (CAPEX) – which mainly refers to network construction expenditure – and OPerating Expenditure (OPEX) – that covers the cost needed to make the network work. Therefore, economical attention is particularly required for new RAN deployment. And vRAN can introduce considerable savings. Indeed, the multiplexing gain let the CAPEX being reduced, as is the number of BBUs. Then the possibility of switching of BBU Pools during low traffic load periods allows to decrease the OPEX, also.

Increased of throughput The 5G radio access scheme, as well as the LTE one, is based on OFDMA and time and frequency resources are dynamically assigned. Moreover, BSs are overlapped in the time-frequency domain, meaning that they use the same resources (reuse factor of 1). Consequently, some techniques are needed in order to bound the inter-cell interference, regardless the adopted RAN architecture. The Inter-cell Interference Coordination (ICIC) mechanism allows a user to report any high interference to the BS; this latter will coordinate with those nearby in such a way as to limit the usage of the interfered resources. But this mechanism adds some constraints to the resource controllers and schedulers, and still the control channels – whose allocation is fixed – interfere. That is why enhanced ICIC (eICIC) was introduced. This last mechanism introduces Almost Blank Sub-frames (ABSs) with the aim of sending without interference relevant information, and requires coordination among BSs. Coordinated Multi-Point (CoMP) is the most relevant collaborative technique that deals with, and at best exploits, interference. The idea behind is that different BSs, if finely coordinated, can cooperate in order to serve the same user. In the most basic implementation, only one base station exchanges data with the user, while the other ones do not interfere. In the most structured version, instead, adjacent cells send the same data to the same user, who can coherently sum the received signal and increase the experienced SINR. These mechanisms require so tight and fast inter-cell synchronization that vRAN can provide centralizing in one unit the digital processing and control tasks, i.e by pooling the BBUs.

Decrease of delays The handovers clearly become less time-consuming if done between cells served by the same BBU Pool.

Network maintenance facilitation In case of failures, the human intervention is required just at the BBU Pool, and not over different baseband units. Moreover, failure frequency may be reduced, since traffic load peaks decrease.

Network upgrades facilitation By virtualizing the BBU Pool, network upgrades can be deployed just by update the running software instead of updating the hardware.

2.3 Towards a computation-aware RAN

Although the benefits of virtualized-RAN are undeniable, and despite the study and research that has been devoted on it over the last years, still some open challenges exist. One of them is the dimensioning and the management of the computational capabilities – and consequently, as it will be explained, of the radio resources.

Current RANs work ignoring the problem of limited computational resources, causing what in [7] is referred to as "*computational outage*". This last happens whenever a decoding failure occurs due to the violation of a timing constraint – mobile networks are subjected to tight timing constraints, and so processing tasks must be performed within given time intervals – rather than due to insufficient channel conditions. This causes a wastage of resources and a degradation of the performance. In light of the above, the need of computation-aware RAN design arises.

To counter this problem, the studied solution in this thesis work foresees two main entities: a *CPU policy* and a *Radio policy.*

CPU policy

The total BBU Pool computational capacity must be shared among all the RAPs under its domain. The CPU policy task is to assign to each RAP a portion of its total computational capacity, in order to serve their demands and support their requirements in terms of quality of service.

Radio policy

Given the assigned computational capacity, also radio resources must be allocated. More precisely, each connection requirements have to be met by choosing a proper Modulation-and-Coding Scheme (MCS). This choice is strictly related to the first one.

2.4 Relationship between radio and computing resources

The relationship between radio and computing resources is far from trivial.

First of all, cellular networks exploit adaptive MCS choice solutions. More precisely, in current RAN implementations the MCS choice is made based on UE's channel condition, which is expressed and communicated to the BS through a parameter called Channel Quality Indicator (CQI). For instance, LTE involves the use of 27 different MCSs, each of which is identified by an index $I_{MCS} = \{1, \ldots, 10\}$ and characterized by different modulation order and transport block size – the code rate is consequently defined –, once that a given number of Resource Blocks (RBs) is assigned to the connection. Three modulations are used: QPSK ($0 \leq I_{MCS} \leq 9$), 16-QAM ($10 \leq I_{MCS} \leq 16$) and 64-QAM ($17 \leq I_{MCS} \leq 26$). The radio access scheme adopted by 5G is quite similar, since both 4G and 5G rely on OFDMA technology. Obviously, by increasing the MCS index, the number of transmitted information bit increases too, making the connection weaker.

In [8] and [9] the relationship between computational and radio resources is explored in detail and modelled. It is well known that users associated with higher MCS incur in heavier instantaneous computational load, but it turned out that their relationship is all but not linear.

The parameters involved in MCS choice are:

- CPU time allocation;
- adopted MCS;
- radio channel condition (expressed in terms of SNR);
- traffic load;
- experienced throughput – as performance metric.

Under the hypothesis of high SNR, i.e. with radio channel conditions that do not bound the set of eligible MCSs, the highest MCS would have been always preferred if the CPU time allocated to the connection would not have been taken into account, since it maximizes the user's experienced performance. But the computational effort required to decode a sub-frame increases with the number of information bits. It happens that, if the BS's load is considerable, then the assigned computational capabilities may be not sufficient to serve all the connections with the chosen MCS. This leads to severe CPU outages, breaking down system performance. Consequently, the CPU time allocation must represent a constraint for the Radio policy. It is preferable lowering the demands on the throughput whenever the CPU resources are scarce.

Then, the computational effort also depends on the number of iterations that the decoder must perform, which instead depends on the SNR. Considering so scenarios represented by worst radio channel conditions, the eligible MCS are bounded by the decreasing CQI – which is already taken into account by current RAN implementations – and by the increasing computational strain, also. If the same MCS is adopted in two different scenarios, then the computational load will be higher for the connection with lower SNR.

The last parameter, which also plays an important role in the radio choice mechanism, is the traffic load of the RAP. Indeed, the resource requirements change based on the amount of information that the RAP need to manage. For instance, in a scenario characterized by low traffic load, the BS do not require much CPU effort since it can afford not to have high demands on the MCS to be adopted. A lower modulation order can be sufficient to send all the information within the defined time interval. On the other hand, whenever the traffic load is high, then it can be preferable for the user to adopt a higher MCS, which may cause some drops, but achieving a higher overall throughput.

In light of the above, it is clear that the relationship between computational and radio resources is far from trivial and linear. Nevertheless, the vRANs need to jointly consider both the resource types. An analytical model may result tuned for specific network conditions, with loss of generality. That is why it has been chosen to study and adopt a model-free solution in order to solve the dimensioning and the management of the computational capabilities issue.

Chapter 3

Machine Learning: fundamental concepts

Machine learning (ML) is a sub-branch of artificial intelligence (AI) concerned with exploiting past experience and data to make intelligent decisions, build coherent and detailed models, recognize complex patterns inside the data. It is a promising technology that has been out for more than 50 years, but which is experiencing strong use only in the last 15 years. In particular, it is getting a lot of interest in the telecommunication field, both in research and market, since it allows to make predictive the management of the network. In [10] ML is formally defined as *"Machine learning is programming computers to optimize a performance criterion using example data or past experience. We have a model defined up to some parameters, and learning is the execution of a computer program to optimize the parameters of the model using the training data or past experience. The model may be predictive to make predictions in the future; or descriptive to gain knowledge from data, or both"*. ML can be so exploited either when humans can not extract the essential information from a system or a problem, or whenever its knowledge can not be formally defined, e.g. voice recognition. And the basic idea behind ML paradigm is to compensate this lack or inability with data, believing that this latter is ruled by a model. And the goal of ML is exactly to extract a good approximation for this model, catching its patterns and regularities. Moreover, the success of ML was strengthened by great ability to cheaply store a large amount of data. The applications of ML, as well as its algorithms, are innumerable, so much so that the choice of the right algorithm to implement forms part of any machine learning program. Even so, three basic macro categories of ML have been identified:

- **Supervised Learning**
  In supervised learning the training data is assumed to be labelled, and the aim of the relative ML algorithms is to learn the mapping, the relationship between data and labels, also named ground truth. Digit recognition is a common example of supervised learning, in which the input data are handwritten images, and the output is a class that categorizes the input as a digit.

- **Unsupervised Learning**
  In unsupervised learning there is not any ground-truth, any supervisor. The ML algorithm must autonomously grasp the essential information and patterns of the input data. The most common method used in unsupervised learning is cluster analysis, which has the goal of segmenting the dataset in groups (clusters), in such a way that this latter is formed by samples with shared features, only.

- **Reinforcement Learning**
  The goal of reinforcement learning problems is to make a *software agent* (the RL learner) learn an *action-choice policy* that maximizes a *reward function*. The learning method is a trial and error one. More precisely, the agent can learn the optimal policy by interacting with the environment and receiving feedbacks, both positive and negative. Reinforcement learning finds use in problems for which an explicit solution is not available.

In this chapter it is presented an overview of the concepts and the deriving algorithms that have been exploited and deployed in this work thesis.

3.1 Artificial Neural Networks

Artificial neural networks (ANNs), conventionally simply called neural networks (NNs), are computing models composed by artificial "*neurons*". From the mathematical point of view, NNs implement a huge multidimensional non-linear function by which complex models can be approximated.

3.1.1 Artificial Neuron: the neural network elementary unit

The NN's elementary unit is generally named artificial neuron, inspired by the biological ones that bring into being the neural processing in the brain.

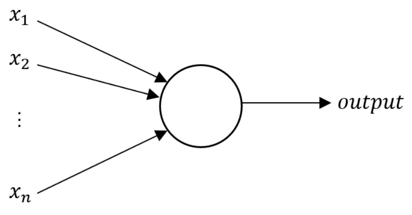


Figure 3.1: Artificial neuron

In the 1950s the scientist Frank Rosenblatt introduced the *perceptron*, which is the elementary and pioneering neuron model. A perceptron, which is depicted in figure 3.1, is an entity that takes as input several binary quantities and produce a binary output, namely it takes a decision based on the processed values. More precisely, the output is computed by the following rule:

$$output = \begin{cases} 0 & \text{if} \quad \sum_j w_j x_j \leq threshold \\ 1 & \text{if} \quad \sum_j w_j x_j > threshold \end{cases} \tag{3.1}$$

in which w_j are real numbers that express the weights for the inputs. The *threshold* instead indicates the difficulty with which the neuron outputs 1. Dropping the threshold means indeed that the neuron more easily can return 1, namely *fires*.

Generally, the expression 3.1 is simplified and rewritten by introducing the dot product and the bias b, as the following:

$$output = \begin{cases} 0 & \text{if} \quad w_j \cdot x_j + b \leq 0 \\ 1 & \text{if} \quad w \cdot x + b > 0 \end{cases} \tag{3.2}$$

in which w and x are vectors whose elements are, respectively, the weights and the inputs of the NN, so that $w \cdot x = \sum_j w_j x_j$, and $b \equiv -threshold$. That means that the bias indicates the ease, not the difficulty as the threshold, with which the neuron outputs fires.

With the perceptron, therefore, the concept of an entity that can produce decisions by weighing some elements arises, but in the majority of ML applications it is inapplicable and unexploitable. The reason for this limit is given by the non-linearity of its function: in order to design a learning model, it is required that small changes in neuron's parameters cause a proportional change in the output. A fully explanation for this will be given successively in section 3.1.3. That is why other models for the artificial neuron have been introduced.

The most common of these is the sigmoid neuron. Its model and parameters – weights and bias – are the same as the perceptron ones, and so it can be depicted as in figure 3.1. But considerable differences are introduced. At first, both the input and the output values are not binary anymore, but real numbers. Then the output's rule is redefined as $\sigma(w \cdot x + b)$ where:

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}}$$

As it is defined, the sigmoid neuron approximates the perceptron. Figure 3.2 shows the similarity of their behaviours.
It follows that what really matters is the shape of the output's law, not the exact expression. And consequently also other functions $f(\cdot)$, rather than the sigmoid one, can be introduced. The Rectified Linear Unit (ReLu) is just another example that deserves to be mentioned. $f(\cdot)$ will be generally referred to as *activation function*[1].

3.1.2 Neural network architecture

It is difficult and very unlikely to think that the artificial neuron alone can approximate complex non-linear models or functions. For this reason, neurons are commonly exploited in groups, that is, connected to each other and forming neural networks.

[1]Note that if $f(\cdot)$ had been a step function, then the perceptron would have been obtained.

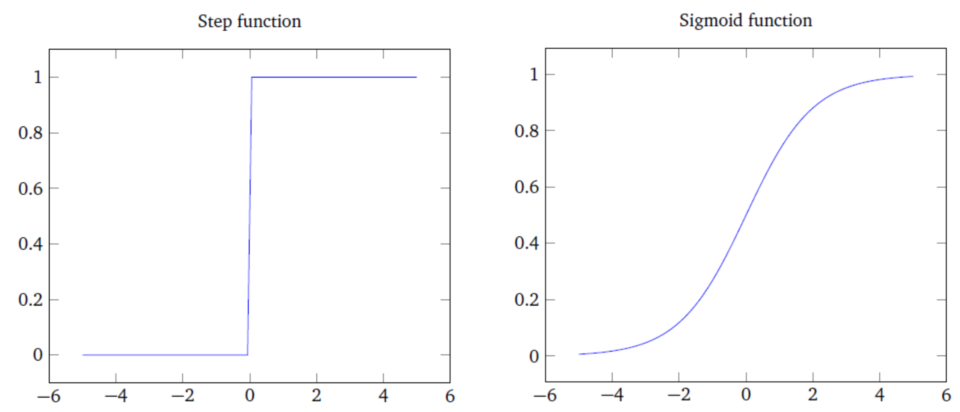


Figure 3.2: Perceptron and Sigmoid neuron: activation functions comparing

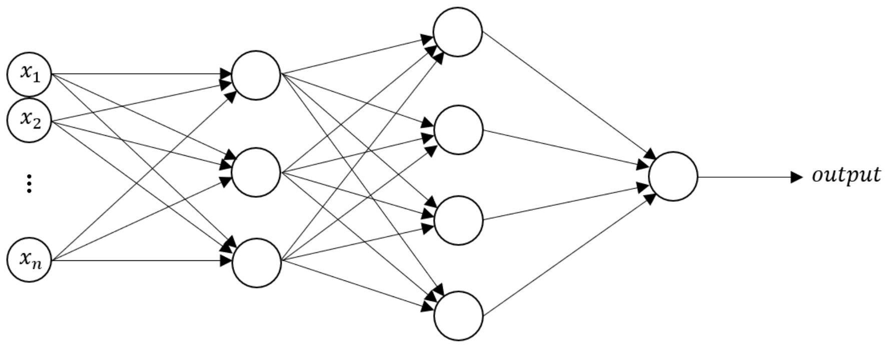


Figure 3.3: Fully-connected feed-forward network

In the trivial example of figure 3.3, neurons are grouped in columns, that are called *layers*. Neurons of a given layer take the output of the previous layer as input, and pass their outputs to the following one. The first layer is called *input layer*, the last one *output layer*, and the remaining ones *hidden layers*. Intuitively, the second layers can take decisions at a more complex level than the first one. Consequently, a NN formed by many layers of neurons can be tuned so to approximate sophisticated functions.

The example in figure 3.3 showed what is referred to as *feed-forward NN*, that is a network in which there are not loops, or equivalently, whose connections do not form any cycle. Moreover, it is a *fully-connected NN*, since each neuron in layer n is connected to all neurons in the layer $n+1$.

There are other two basic NN architectures, which are the *convolutional* and the *recurrent* ones. The first ones are particularly suited for two-dimensional inputs, like images, but still they do not present any loop, contrarily to the recurrent NNs which find their applications in dynamical systems. These two families of NN are not deepened in this work, even if the learning methods that will be introduced below are hold for NNs in general.

3.1.3 Gradient descent algorithm

Gradient descent is an algorithm that, when applied to NNs, can automatically tune its parameters so that it learns the model of the input data. This algorithm leverages on the idea that to a small change in the parameters of the NN corresponds a proportional change to its output and vice versa. And this was the reason of perceptron problems that has sprung the design of smooth activation functions. Consequently, we can modify the network so that it behaves in the desired manner, i.e. it outputs the expected values corresponding to a given input.

The first step towards the design of the learning algorithm is the definition of a criterion, which will be equivalently called *cost*, *objective* or *loss function.* This latter is a metric of how far the answer of the neural network is with respect to the desired behaviour.

The most commonly used and simple cost function is the quadratic one, which is also referred to as Mean Squared Error (MSE).

$$C(w,b) = \frac{1}{2n}\sum_{x} \|y(x) - a(x,w,b)\|^2 \tag{3.3}$$

It is worth to note that the loss function only depends on the parameters of the NN, w and b. $a(x,w,b)$ instead is the vector of NN outputs when the set of data x is given as input: $a(x,w,b)$ is required to approximate $y(x)$, namely the desired outputs. $C(w,b)$ is a non-negative function that tends to zero whenever $a(x,w,b)$ approaches $y(x)$.

Consequently, a good learning process is required to tune the weights and biases of the NN so that $C(w,b) \approx 0$. To put it plainly, the learning algorithm must minimize the cost function, looking for its global minimum.

Computing analytically the minimum is obviously unattainable, but the derivative can be relied. More precisely, if the loss function was a function of $v = (v_1, \dots, v_m)^T$, then a change $\Delta v = (\Delta v_1, \dots, \delta v_m)^T$, would have led a change ΔC on the cost function approximately equal to:

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \dots + \frac{\partial C}{\partial v_m}\Delta v_m = \nabla C \cdot \Delta v \tag{3.4}$$

where the gradient vector ∇C is defined as:

$$\nabla C = (\frac{\partial C}{\partial v_1}, \dots, \frac{\partial C}{\partial v_m})^T$$

As a consequence, chenges in v and changes in C are related by the gradient vector. Gradient descent algorithm defines a way to tune Δv in order to trigger a negative change in ΔC, namely a decrease of the loss function. If indeed it is supposed to pick

$$\Delta v = -\eta \nabla C \tag{3.5}$$

where $\eta \geq 0$ is a parameter that will be named *learning rate*, then the corresponding variation of the loss function would be:

$$\Delta C \approx \nabla C \cdot \Delta v = -\eta \nabla C \cdot \nabla C = -\eta \|\nabla C\|^2$$

Being $\|\nabla C\|^2 \geq 0$, it holds $\Delta C \leq 0$.

Gradient descent algorithm introduces so the concept of update rule, whereby the parameters on which the loss function depends are updated.

$$v \to v' = v + \Delta C = v - \eta \nabla C$$

Such a rule is significant under the hypothesis that the approximation 3.4 holds, that is to say when Δv is limited. Therefore, if 3.5 is assumed, then the learning rate has to be small enough. Simultaneously, η should be large enough to let the update rule be not insignificant. Gradient descent algorithm applies this updating several times, reducing the loss function from time to time and hopefully reaching its global minimum. Every iteration is called *epoch*.

If gradient descent is made used of in NN learning process, then the update rule becomes:

$$w_k \to w'_k = w_k - \eta \frac{\partial C}{\partial w_k} \tag{3.6}$$

$$b_l \to b'_l = b_l - \eta \frac{\partial C}{\partial b_l} \tag{3.7}$$

The beauty of gradient descent algorithm are its effectiveness, since in practice it often works extremely well, and its particularly low computational load. Other solutions have been investigated, but they all are more complex, since they involve the computation of the second derivative.

However, gradient descent algorithm comes with several challenges in ML field. One of them is the following: since the total loss function (3.3) is the average of the losses due to each of the m input data, then also ∇C has to be averaged over the same number of components. More precisely, it follows:

$$\nabla C = \frac{1}{n} \sum_x \nabla C_x \tag{3.8}$$

And if the number of training data is too large - as in ML always is - then the computation of ∇C becomes too heavy, and the learning process slow. This is the reason why *stochastic gradient descent* algorithm has been introduced. The idea behind this latter is not to exploit the whole training dataset together, but to use random portions of samples that will be named *mini-batches* until the entire dataset is covered. If M is the number of mini-batches and ∇C_{x_j} the divergence vector of C computed over the j-th mini-batch, then 3.8 can be approximated as:

$$\nabla C = \frac{1}{n} \sum_x \nabla C_x \approx \frac{1}{M} \sum_{j=1}^{M} \nabla C_{x_j} \tag{3.9}$$

And also 3.6 and 3.7 must be modified:

$$w_k \to w'_k = w_k - \frac{\eta}{M} \sum_{j=1}^{M} \frac{\partial C_{x_j}}{\partial w_k} \tag{3.10}$$

$$b_l \rightarrow b'_l = b_l - \frac{\eta}{M}\sum_{j=1}^{M}\frac{\partial C_{x_j}}{\partial b_l} \tag{3.11}$$

Note that the approximation 3.9 is acceptable if and only if the size of the mini-batches is not so restricted.

But the biggest challenge that gradient descent involves is the computation of the gradient of the cost function. In this regard the *backpropagation algorithm* was introduced.

3.1.4 Backpropagation algorithm

The backpropagation algorithm has collected its particular attention in 1986 with the publication of [11], even if introduced for the first time in the 70s. Backpropagation algorithm responds in an easy and effective way to the need to compute the derivatives of the cost function with respect to the weights ($\partial C/\partial w$) and the biases ($\partial C/\partial b$) of the NN.

At first, it is necessary to introduce some notational conventions.

- With w^l_{jk} it will be identified the weight of the connection between k-th neuron of $(l-1)$-th layer and j-th neuron of l-th layer.
- With b^l_j it will be identified the bias of the j-th neuron of l-th layer.
- With a^l_j it will be identified the activation of the j-th neuron of l-th layer.

The output of the j-th neuron of l-th can be so computed as:

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

Introducing the matrix notation, so defining for each layer the weight matrix $w^l = [w^l_{jk}]$, the bias vector $b^l = [b^l_j]$ and the activation vector $a^l = [a^l_j]$, it can be written:

$$a^l = \sigma(w^l a^{l-1} + b^l) \tag{3.12}$$

where $\sigma(\cdot)$ acts over each element of its input (elementwise operation).

The expression 3.12 directly relates the output of one layer to the outputs of the layer before and can be further simplified as:

$$a^l = \sigma(z^l)$$

where $z^l = w^l a^{l-1} + b^l$ is a vector with components $z^l_j = \sum_k w^l_{jk}a^{l-1}_k + b^l_j$ that will take on an important role in backpropagation algorithm.
Before introducing the heart and soul of the algorithm, some assumptions have to be made.

The first one regards the loss function, that has to be written as an average of the losses related to each input data.

$$C = \frac{1}{n}\sum_x C_x$$

If the MSE is assumed as loss function and $l = L$ indicates the output layer – L is so the total number of layers –, then it follows:

$$C_x = \frac{1}{2}\left\|y(x) - a^L\right\|^2 = \frac{1}{2}\sum_j (y(x)_j - a^l_j)^2 \tag{3.13}$$

Such an assumption is required since the backpropagation algorithm will define a way to compute the partial derivatives of C_x with respect to the parameters of the NN, *i.e.* $\partial C_x/\partial b$ and $\partial C_x/\partial w$. Then, 3.8 will be used in order to recover the partial derivatives of the total loss function.

The second important assumption provides that the loss function can be expressed in relationship of the NN's output. This supposition has been already introduced in 3.13 for the MSE cost function.

Backpropagation algorithm provides a way to compute the partial derivatives of the loss function in relationship to an auxiliary parameter, which is called *error*:

$$\delta^l_j \equiv \frac{\partial C}{\partial z^l_j}$$

δ^l_j identifies the error in j-th neuron of l-th layer. The error can be interpretated as a measure of the unsuitability of the neuron: if the error is large, then the subset of NN parameters involved in z^l_j can be modified so to have a consistent improvement of the loss function. Contrary, if the error δ^l_j is bounded, then the j-th neuron of l-th layer is almost optimal.

Backpropagation algorithm introduced four key equations which are presented, without demonstration, in the following.

Equation BP1: error at the output layer

$$\delta^L_j = \frac{\partial C}{\partial a^L_j}\sigma'(z^L_j) \tag{3.14}$$

In $BP1$ two terms are involved: the first one is a measure of how the cost function changes in relationship to the output layer activation functions, while this latter vary at its input z^L_j as expressed by the second term.
If BP1 has to be expressed in matrix-form, then it can be rewritten as:

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{3.15}$$

where with $\odot$ it is indicated the elementwise product (*Hadamard product*).
Note that each term if BP1 is easily computable once that the loss function is defined and the NN's input data is obviously given.

Equation BP2: error at layer *l* (δ^l) in terms of error at layer *l+1* (δ^{l+1})

$$\delta^l = ((w^{l+1})^T\delta^{l+1}) \odot \sigma'(z^l) \tag{3.16}$$

BP2 allows, thank to the term $(w^{l+1})^T$, to *backpropagate* the errors.
Note that by jointly exploit equations BP1 and BP2, it is possible to compute the

error in any neuron of any layer. What is missing at this point is the relationship between the errors and the partial derivatives of the loss function.

Equation BP3: rate of change of the cost with respect to any bias

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{3.17}$$

BP3 tells that by computing the errors, also the partial derivative of the cost function with respect to the biases will be given, since they are equal.

Equation BP4: rate of change of the cost with respect to any weight

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l \tag{3.18}$$

Also the partial derivatives of the cost function with respect to the weights can be simply obtained by means of BP4 and the easily computable a_k^{l-1}.

In light of the four equations described above, the backpropagation algorithm includes the following steps:

Feedforward step
Given the input x, feed the NN with weights w^l and biases b^l. More precisely, compute for each layer $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma z^l$ and the deriving output layer error using 3.15.

Backpropagation step
Backpropagate the errors at the output layer with 3.16 and derive the resulting partial derivatives with 3.17 and 3.18.

Neural Network parameters updating
Once the partial derivatives are give, update the parameters of the NN using the equations provided by stochastic gradient descent: 3.10 and 3.11.

The backpropagation algorithm lays the foundation for any neural network learning process. Clearly, many other details and challenges would require to be discussed, such as, at first, the choice of activation or cost functions, data pre-processing or hyperparameters (*e.g.* learning rate and batch size) tuning. But as this thesis work is not intended to analyse all the details of NN and ML, they are here skipped.

3.2 Reinforcement Learning

Reinforcement learning (RL) is the most advanced ML paradigm and it is aimed at learning a control policy, namely a strategy whereby a software agent in a given environment decides what action to perform in order to maximize a reward.

The method chosen by RL through which the learner must discover the best control strategy is a *trial and error* one: the agent must evaluate the actions by taking them, it must learn from its past experience.

The differences between RL and the main other branches of ML are substantial. On the one hand, differently from supervised learning, in RL there is not a set of examples of desired behaviour, namely a set of data paired with the corresponding best action that should have been taken (the label). The agent must understand singlehandedly what are the best actions to take and in what context. On the other hand, contrarily from unsupervised learning, the agent is not tasked with figuring out the hidden structure behind the provided samples, but it must solve a decision-making problem bounded to the maximization of the reward. Obviously, extrapolating the hidden structure of the examples may be essential, but not enough.

Considering examples is a very efficient way to get a better insight of RL. One of the most commonly chosen examples is the chess game: each player, based on the current position of the chess pawns, has to make a decision about what move to make. The move choice depends on the actual pieces placement, but also on the immediate consequences it entails *e.g.* capture an opponent's piece, and on the anticipations about the following moves that the player can plan. Once the move is performed, clearly the chessboard setup is turbated. Moreover, in order to improve his skills and understand the dynamics of the game, each player must play more and more games.

From this example some aspects of RL are clear. At first, the agent can interact with the environment: he observes a state (the chess pawns position) and decides an action (a chess pawns move) that modifies the environment. This latter instead rewards or penalizes (with a low reward) the agent's choice. i.e. the agent wins or loses the game, captures or has a piece captured. But by modifying the environment, the agent influences also subsequent states and can improve or deteriorate the chances of obtaining high rewards.

This actor-environment interaction is illustrated in figure 3.4.

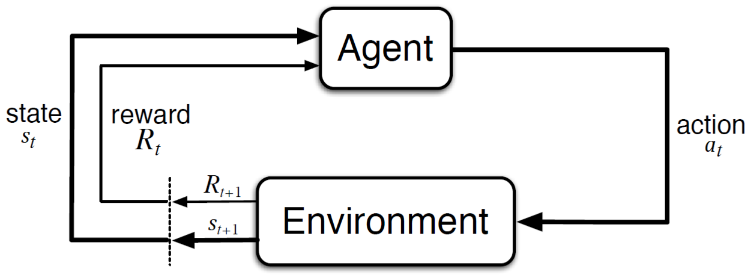


Figure 3.4: The agent-environment interaction

3.2.1 The basic elements of RL

The fundamental elements of RL are basically four [12]: the *policy*, the *reward signal*, the *value function* and, arbitrarily, the *model*. In order to formally describe these lasts, it is necessary to properly define the state and action spaces.

The set of environmental states S, which has dimension N, is defined as $S = (s_1, \ldots, s_N)$.

Then, the set of actions A, which has dimension K, is defined as $A = (a_1, \ldots, a_K)$.

The subset of actions that can be selected in state s is instead indicates with $A(s)$.

Whenever the agent performs an action $a \in A$ from state $s \in A$, then the environment moves in the following state, $s' \in S$, with a certain probability. The notion of *transition* $T : S \times A \times S \to [0,1]$ is related to this change of state.

- **Policy**
  The policy, which is the goal of RL and that entirely determines the agent's behaviour, is the function that maps the observed state $s \in S$ into an action $a \in A$ (or, more accurately, into $a \in A(S)$). In particular, two kinds of policies can be identified.
  A *deterministic policy* directly selects the action to be performed: $\pi : S \to A$.
  A *stochastic policy* outputs instead a distribution of possible actions: $\pi : S \times A \to [0,1]$.
  The probability associated to the pair *(s,a)* will be identified with $\pi(a|s) \in [0,1]$.

- **Reward signal**
  The reward is what allows the agent to learn the policy, its evaluation criterion. The task of the policy is indeed to obtain the highest possible reward over the long-run, namely after the reach of the final state (in case of a chess game, the end of the match) or a considerable number of steps. An optimal policy leads so to the maximum reward.
  At each step, the agent observes the current state s_t, takes action a_t and receives a reward signal R_{t+1} from the environment that ends in state s_{t+1}. If the received reward is low, then maybe the policy can be improved.
  The reward signal R associated to the *state-action-next state* triple (s, a, s') can be analytically defined as: $R : S \times A \times S \to \mathbb{R}$.
  At this point it can be defined the *Markov Decision Process* (MDP) *framework* as the tuple $< S, A, T, R >$. The Markovian property holds if the result of an action is not affected by the previous ones, but only depends on the actual state. In other words, at each step the agent can choose the optimal action taking into account only the current state, without keeping track of its past choices.

- **Value functions**
  But the immediate reward is not enough to evaluate a policy. Indeed, a criterion that also evaluates future steps is needed.
  Going back to the example of before about chess, a player can decide to lose a piece, thus receiving a low immediate reward, but forcing the opponent to break its defence and so creating an opportunity for victory. With the first move, therefore, the player created the condition to end up in a convenient state, with a high associated reward. But clearly the opposite situation can also happen.
  These examples can be generalized in the concept that the effect of an action with respect to the goal can be much delayed. Under the light of this idea, the notions of *cumulative reward* the *value functions* were introduced.
  The cumulative reward at the t-th time step, G_t, also called *return*, can be computed in the simplest case as sum of the individual rewards over the next time steps:

$$G_t = R_{t+1} + R_{t+2} + \cdots + R_T$$

But this approach can be followed only if the return does not diverge, namely if the number of time steps is finite. It is the case of chess game, for example: the match must end sooner or later.
But commonly in RL it is introduced the concept of *discounted reward*:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \tag{3.19}$$

The idea behind the discounted reward is that delayed rewards worth less than immediate ones, so they must be penalized by a factor γ^k, where k is the delay expressed in terms of time steps. If $\gamma < 1$, then the expression 3.19 certainly converges assuming R finite.
The *state value function* is then defined as the amount of return that the agent can expect, that is, it indicates the *value* of the current state. Obviously, the state value function depends on the current policy and it can be denoted as:

$$v_\pi(s) = \mathbb{E}_\pi[G_t|s_t = s] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, s_t = s\right]$$

Similarly, the *state-action value function* can be defined as the expected return starting from state s and taking action a:

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

A policy that maximizes a value function, and not the reward, is the real objective of RL.

$$v_*(s) = \max_\pi v_\pi(s) \quad \forall s \in S$$

$$q_*(s,a) = \max_\pi q_\pi(s,a) \quad \forall s \in S, \forall a \in A(s)$$

But while the reward is given, as it is given directly by the environment, the value function must be estimated in advance by the agent, before taking any action.
A relevant property of these value functions is a recursive relationship called *Bellman equation.* This latter shows how the value of a state is related with the values of the succeeding ones and represents the basis for a large variety of RL algorithm. The Bellman equation is defined for state value functions as:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')] \tag{3.20}$$

where $p(s',r|s,a) = Pr\{R_{t+1} = r, s_{t+1} = s'|s_t = s, a_t = a\}$

- **Model**
  The last basic element of RL is the model for the environment, namely for the MDP.
  The algorithms that exploit this entity are called *model-based algorithms.* The others instead are named *model-free algorithms* and rely on interaction with the environment. More precisely, the agent is forced to explore the environment in order to gather information about the MDP.

In the following two sections the main reinforcement learning mechanisms are briefly investigated. Their details are not studied in deep in this work thesis since the solution that will be proposed in chapter 4 will rely on a novel algorithm called *Actor-Critic*. This latter will be instead introduced in section 3.2.4.

3.2.2 The model-based learning mechanism: a brief overview

Whenever a perfect model for the MDP is given, the learning of the optimal policy takes the name of *Dynamic Programming* (DP). This class of solutions is often unfeasible both because RL mainly deals with model-free solutions and because DP is computationally complex. But still DP is relevant, since it lays the foundation for model-free solutions.

- *Policy iteration methods*
  In policy iteration methods two different phases can be identified: the *policy evaluation* and the *policy improvement* steps.
  The policy evaluation phase is tasked with the computation of v_π for each state s. One way to proceed is to exploit the Bellman equation (3.20), which involves solving a system of $|S|$ expressions and $|S|$ unknowns – the $v_\pi(s), \forall s \in S$, while all the other terms are easily accessible by the MDP model. But this would carry a huge computational burden, and that is why an iterative approach has been defined: starting from an initial approximation $v_0(s)$, the evaluation of the state value function can be updated by:

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')] \tag{3.21}$$

  The approximations $v_k(s)$ converges to the real state value function as $k \to \infty$.
  The policy evaluation step is required in order to look for better policies. This latter is the goal of policy improvement step. The idea behind this phase is to compute the state-action value function for each state and for each action, and then compare the result with the policy evaluation's output: if the state-action value is for some states higher than the value-function, then the policy can be improved, so a new policy π_{k+1} can be computed. Otherwise, the policy is already optimal.
  If a deterministic policy is looked for, then it can be updated as:

$$\pi_{k+1}(s) = \arg\max_a q_\pi(s, a) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \tag{3.22}$$

  But this approach can be generalized for stochastic policies, also.
  Summarizing, policy iteration methods start from a policy π_0, which is at first evaluated with 3.21, and then improved with 3.22. These steps can be iterated.
  For finite state and action spaces – finite MDPs –, policy iteration allows to determine the optimal policy after a finite number of iterations.

- *Value iteration methods*
  In value iteration methods, the policy evaluation and policy improvement steps

are not completely separated as for policy iteration methods. The basic idea of value iteration methods is that it is not strictly necessary to perfectly evaluate the state value function of the policy before going to the policy improvement step. In particular, in value iteration methods the policy evaluation is stopped after just one step.

3.2.3 The model-free learning mechanism: a brief overview

As already anticipated, with model-free algorithms are meant all the learning methods that do not rely on a complete model for the environment. Hence, the agent is forced to remedy this lack, namely to gather statistical information about the environment, by interacting with it, by taking actions and estimating the corresponding value functions.

In model-free learning procedures several options can be chosen [12].

At first, the algorithm could learn directly the transition and reward model and then apply the DP procedures (*indirect RL*) or it could learn directly the value functions (*direct RL*), without modelling the MDP. This last option is commonly the adopted choice.

The second point is related to how to deal with what is referred to as *credit assignment problem*. More precisely, since the effect of an action can be considerably delayed, then how to give credit or penalize a past action becomes an issue. An option is to wait the environment to end up in the final state, meanwhile keeping track of past choices. Another one is called *temporal difference learning* and requires the agent to update the value functions estimates based on the immediate received reward and the estimated values of the next states (*bootstrapping*). This approach is similar to the value iteration one.

Another important issue related to model-free solution lies on the fact that the agent definitely needs to sample the environment, to explore new state-action pairs, in order to look for the best choice. But the exploration has also to be moderated, so the agent can exploit its current knowledge. This issue is known as the *exploration-exploitation trade-off problem*, and generally independent *exploration mechanisms* are adopted.

The most basic and popular temporal difference learning algorithms are *TD(0)*, *SARSA* and *Q-Learning*.

- **TD(0)**
  The update rule exploited in TD(0) is the following:

  $$v_{k+1}(s) = v_k(s) + \alpha\Big(r + \gamma v_k(s') - v_k(s)\Big) \tag{3.23}$$

  which differs from DP policy improvement (expression 3.22) since only the observed iteration is taken into account, not the weighted average of all possible successor states. α is the learning-rate.

- **Q-Learning**
  Q-Learning is a method that directly learn the state-action value function. Every time that the agent performs an action, the relative state-action value function estimate is updated, according to an update rule that can be derived from 3.23.

  $$q_{k+1}(s_t, a_t) = q_k(s_t, a_t) + \alpha\Big(r_t + \gamma \max_a q_k(s_{t+1}, a_t) - q_k(s_t, a_t)\Big) \tag{3.24}$$

Q-Learning is an *off-policy* algorithm, namely an algorithm that is insensitive from the exploration policy.

- **SARSA**
  SARSA is the on-policy version of Q-Learning, since it still tasked with estimate the state-action value function, but it depends on the current exploration method. SARSA works particularly well for *non-stationary* systems, under the hypothesis that the exploration factor is incrementally attenuated. Its update rule is the following:

$$q_{k+1}(s_t, a_t) = q_k(s_t, a_t) + \alpha\Big(r_t + \gamma q_k(s_{t+1}, a_{t+1}) - q_k(s_t, a_t)\Big)$$

  Note that the max-operator is replaced with the estimate of the state-action value of the next state-action pair (s_{t+1}, a_{t+1}).

3.2.4 Actor-critic Learning

The algorithms previously introduced aimed to estimate either the state or the state-action value functions, on the basis of which to implement the policy, namely directly select the action. All the methods that are founded on this idea are named *action-value methods* [13].
In actor-critic learning the state or action-state value functions are clearly still estimated, but the policy is independently modelled. Namely, the value functions do not directly determine the action to be selected.

The structure that models the policy is called *actor*, while the one associated with the value function is named *critic*. The reason for these names is intuitive: the actor is the entity that selects the action to be executed, while the critic *criticizes* how it performs.

Generally, both the actor and the critic are modelled with two neural networks.

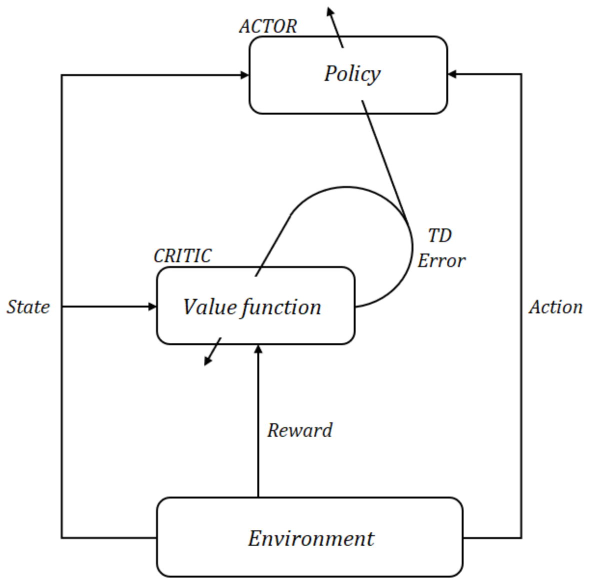


Figure 3.5: The actor-critic architecture

The architecture of actor-critic methods is shown in figure 3.5. Here it can be noticed that, at first, the actor observes a state s from the environment and chooses an action a to be performed. Based on this latter, the environment outputs a reward signal R that is taken as input from the critic, together with the state. So the critic can made its estimate for the value function and update it computing the *TD error* as a function of the received reward. The TD error is also given to the actor as feedback for the chosen action. More precisely, since the the TD error is defined as $\delta_t = R_{t+1} + \gamma v_t(s_{t+1}) - v_t(s_t)$, then a positive error would strengthen the selection of the chosen action in state s; instead a negative error one would clearly weaken it.

Many details for actor-critic solutions will be given in 4, but here their advantages can be introduced.

- Action selection requires a low computational effort, even if the the action is a continuous value, since the actor directly outputs it, without being forced to search for the best selection through the infinite action space.

- They can learn stochastic policy

Chapter 4

Resource Manager: design

In the following chapter it is proposed the design of a dynamic resource controller for virtualized RANs. As it was argued in chapter 2.4, a model-free solution is resorted in order to deal with the complex relationship between radio and computational resources and to dynamically adapt their allocation to the served access points.

Before introducing the details of the presented solution, it is worth to describe the scenario that will be taken into account in order to procure the dataset required to train and test the solution.

4.1 Simulation platform configuration

In order to define and optimize the design of the resource controller it was decided to take into consideration the simplest of scenarios in which there is only one vRAP to manage, with one stationary connected user. Clearly this represents the first step before being able to consider more complex scenarios, with more users and subsequently more vRAPs.

It was also chosen to consider only the uplink traffic in a 4G LTE link − thus using the *Physical Uplink Shared Channel* (PUSCH) − as this represents the most complex case, in which the user periodically sends feedbacks to the vRAP. The extension to the downlink case is trivial, as the information required by the resource manager is local, *i.e.* it does not need to be exchanged. More precisely, the channel taken into consideration was a 10Mhz channel - the user is therefore assigned 50 RBs - in band 7 (2500 - 2570 Mhz).

Even the generalization to the 5G case is trivial, as the radio access schemes adopted by 4G and 5G are quite similar to each other, since both of them rely on OFDMA technology.

In order to conduct the simulations, it was necessary to implement all three fundamental blocks of the LTE architecture, which are:

- user equipment (UE);
- the 4G access network, which is named *evolved UMTS Terrestrial Radio Access Network* (*E-UTRAN*) and that is composed, in the considered scenario, by just one

eNB;

- the 4G CN, which is named *Evolved Packet Core* (*EPC*).

Starting considering at first UE and eNB, these have been implemented on two different *USRP*s (*Universal Software Radio Peripheral*) which are flexible wireless prototyping platforms belonging to the family of Software Defined Radios (SDRs) and which allow to receive and/or transmit different and arbitrary waveforms at various frequencies. The applications that implement the UE and the eNB entirely in software from the physical layer to the IP layer (therefore the entire LTE stack) are respectively *srsUE* and *srsENB*. The latter two are part, together with *srsEPC*, of an opensource library for experimentation called *srsLTE* and which provides also interfaces to the *Universal Hardware Driver* (*UHD*) − hence supporting the devices belonging to the Ettus USRP family.

Both srsUE and srsENB are computationally burdensome, being responsible for all RF functionalities, conversions, coding, modulation and FFT.

In order to exchange traffic, the USRPs relating to the EU and eNB were connected to each other by means of two coaxial cables: one responsible for uplink communications, the other for downlink communications.

In the deployed testbed, the EPC is co-located with the eNB.

In order to easily limit the computational resources allocated to the eNB, a *Docker*[1] container has been implemented.

Finally, a measurements campaign has been carried out. The aim of this latter was to simulate the behaviour of the described system in a heterogeneous set of contexts, measuring the corresponding performance. Hence, one simulation for each combination of the following parameters has been carried out.

- **cpu_limit** $\in \{1.1, 1.2, 1.3, 1.5, 1.6\}$
  The resource manager has the objective of dynamically distribute the computational resources among the directed vRAPs. The AI-solution for the resource controller has consequently to be trained with a set of samples that reports the obtained system performance relative to a comprehensive set of different CPU allocations. These lasts express the CPU allocated for the srsENB application and can take values in $[0; 8]$ since the CPU that runs srsENB is an *octa-core* one.
  Simulations showed that for values less or equal than 1.1 the obtained performance were insignificant (not a single successful test), while for values greater or equal than 1.6 they were optimal. That is why only the mentioned limited set of CPU allocations has been simulated.

- **max_mcs** $\in \{0, 3, 6, 9, 12, 15, 18, 21, 24, 27\}$
  The second objective of the resource controller is to express, for all the vRAPs, an upper bound to the eligible MCSs. Different MCS bounds were so simulated.

- **tx_power** $\in \{-15, -13, -11, -9, -7, -5, -3, -1, 4, 14\}\, dB$
  In section 2.4 it has been explained that CPU consumptions of the BBU pool mainly

[1]Docker is an open-source project that automates the deployment of applications in software containers.

depend on two factors: the user's experience SNR and traffic load. In order to analyse the impact of the first parameter, a range of UE's transmission powers has been simulated. Indeed, the experienced SNR is proportional to the transmission power. The system noise is instead the white one added by the coaxial cable.

- **traffic_load** $\in \{0.2, 1, 3, 6, 12, 24\}$ *Mbps*
  On the other hand, UE's traffic is CBR (constant bitrate) and $traffic\_load$ expresses its average value.

Each configuration, therefore each combination of the parameters described above, was simulated for 20 seconds.

All the quantities that play an important role in determining the allocation of resources (both radio and computational) were instead sampled at 100 ms intervals, forming what will be referred to as *context information*. The time domain was so divided into *stages* of 20 *seconds* and *monitoring slots* of 100 *ms*. Each monitoring slot is so composed by 200 monitoring slots.

All the measurements were saved in a dataset, which will be exploited in order to drive the learning of the resource manager.

The proposed dynamic resource controller will then operate once for each stage. More precisely, for the entire duration of a stage it collects data about the context information based on which it takes a decision, an action – therefore it expresses a computational allocation and a maximum MCS for vRAP – valid for the next stage.

4.2 Encoder

In our configuration, the first acting entity is an encoder which accomplishes the task of translating the high-dimensional context information into a simpler representation. This step is surely needed since a high-dimensional state space may not be easily handled by the resource manager.

4.2.1 Encoder design

It has been chosen to sum up the context information $x_i^{(n)}$ relative to stage n and vRAP i in three different features sampled after every monitoring slot t. The features are the ones that have the biggest impact on vRAN performance and, consequently, on the management of the resources.

- Mean experienced SNR averaged among all the users connected to the vRAP:

$$\bar{s}_i^{(n)} = \left(\bar{s}_{i,t}^{(n)}\right)_{\forall t \in \{1,\ldots,T\}}$$

- Experienced SNR variance averaged among all the users connected to the vRAP:

$$\tilde{s}_i^{(n)} = \left(\tilde{s}_{i,t}^{(n)}\right)_{\forall t \in \{1,\ldots,T\}}$$

- Total amount of bits available to be transmitted from the vRAP:
$$\delta_i^{(n)} = \big(\delta_{i,t}^{(n)}\big)_{\forall t \in \{1,\dots,T\}}$$
Each active UE communicates its traffic load to the vRAP by means of a parameter called Buffer Status Report (BSR).

The importance of $\bar{s}_i^{(n)}$ and $\delta_i^{(n)}$ in determining the allocation of resources was explained in section 2.4. Even if the simulated scenarios are stationary, it was decided to take into account also the SNR variance $\tilde{s}_i^{(n)}$ since it is a direct metric of user mobility and in dynamic contexts the system performance is expected to decrease to the pairs of CPU allocation and adopted MCS.

Each feature is grouped in a unique vector, and it holds:
$$x_i^{(n)} = \left(\bar{s}_i^{(n)}, \tilde{s}_i^{(n)}, \delta_i^{(n)}\right)$$
As each vRAP is independent from the other ones, it must be considered individually. It has been so chosen to implement, for each vRAP, three different autoencoders, one for each feature constituting the information context. The reason for this option is due to the fact that the autoencoder must exploit the inherent temporal correlation between consecutive samples of a given feature in order to efficiently reduce the context information rather than the correlation between samples of different features. That is why the vectors are processed separately.

An AI solution is required as the context information vectors are complex signals, since they translate users' behaviours. For instance, if the users are stationary, then also the corresponding vectors should present such a property. If the context is instead variable, then the variance among consecutive samples would increase. And especially all the mixed scenarios have to be considered. That is why a pre-defined encoding scheme, like the simplest average or variance, is not feasible: it would lead to a considerable information leak, since it is tailored for a specific case, only. As a consequence, an entity that extrapolates the inherent structure of data is required.

The proposed solution resorts to an unsupervised learning algorithm named Sparse Autoencoder (SAE).

Autoencoders are feed-forward NNs tasked with efficient data coding (representation learning), typically – but not necessarily – for dimensionality reduction. Their architecture imposes a *bottleneck* that forces the input data in a compressed representation, as it is shown in figure 4.1.
As a general rule, in an autoencoder two different symmetric components, namely two feed-forward NNs, are identifiable. The first one is the *encoder*, whose aim is to translate the input data x whose dimension is D_x into a representation $y = e(x)$ of dimension D_y. If $D_y \leq D_x$, then the autoencoder achieves dimensionality reduction. The encoder block is then followed by the *decoder*, which instead has to reconstruct the original data from the encoded one, namely $\hat{x} = d(y) \approx x$. The whole autoencoder is trained so to minimize the reconstruction error $\mathcal{L}(x, \hat{x})$, where $\mathcal{L}(\cdot)$ is an arbitrary loss criterion, as the MSE.

And since the structure that the autoencoder is tasked to look for is complex, as stated before, two choices have been taken.

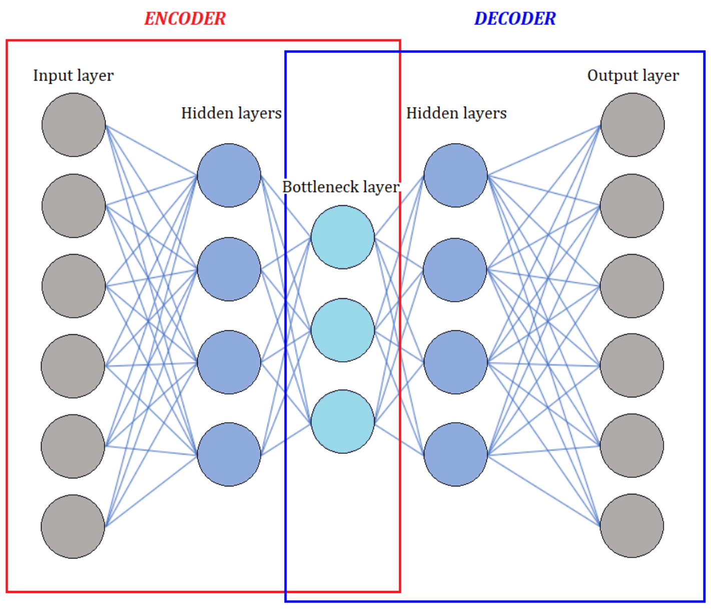


Figure 4.1: Autoencoder architecture

- At first, the implemented autoencoder does not present a single-layer encoder and single-layer decoder (shallow autoencoder), but is a deep one. This may yield better compression compared to shallow autoencoders [14].
- A linear autoencoder (*i.e.* with linear activation function) may not be able to get the complex inherent structure of the data. More precisely, linear autoencoders are recommended for *principal component analysis*, which looks for the main components, *i.e.* the eigenvectors, of the data. But the promise of autoencoders lies in their non-linearity [14], which allows the NN to learn more complex structures. That is why it has been chosen to implement ReLU activation functions in the hidden layers of the deployed autoencoder.

The last regard that has to be discussed about the autoencoder is the implemented optimization criterion – the objective function of the NN. In order to minimize the reconstruction error, a simple MSE loss is more than satisfactory. But to this last, it has been chosen to add a *sparsity penalty* – hence the name *sparse* autoencoder (SAE) –, namely a term that penalizes the activation of the hidden neurons so that only few of them are hearten to fire. The utility and the point of such a sparsity penalty is to encourage the NN to extrapolate the essential knowledge hidden in the information context, avoiding to focus on redundant details. Moreover, the sparsity penalty ensures the autoencoder from *overfitting*[2]. To name but one example in order to get a sense of the sparsity penality, consider an autoencoder with $D_y \geq D_x$. Without any sparsity penalty,

[2]The concept of overfitting is related to any statistical model that exactly or too close fits to the observed data, having an excessive number of parameters with respect to the observations.

the NN could simply copy the input at D_x out of D_y neurons of the encoder's output layer, and filter these neurons in order to reconstruct the original data, without learning the fundamental patterns of the data. But even in representation learning, sparseness may bring important benefits.

The two most common ways to construct the sparsity penalty are *L1 regularization* and *Kullback-Leibler (KL) divergece*. This latter is the one implemented in the proposed solution.

On the one hand, L1 regularization term is proportional to the sum of the absolute values of NN's weights. That means that non-zero coefficients are penalized.

On the other hand, KL divergence is, generally speaking, a measure of the difference among two probability distributions, P and Q. More precisely, the KL divergence of Q with respect to P, indicated as $D_{KL}(P||Q)$, is the measure of information loss when Q is chosen as an approximation for P. This general concept can be specified for SAEs as follows.

Taking into account one single neuron of the NN – the j-th –, it can be considered inactive whenever its output value is close to 0, while it can be though as being active whenever its output is greater. Then its average activation can be computed as:

$$\hat{\rho_j} = \frac{1}{m}\sum_{i=1}^{m} a_j(x^{(i)})$$

where with $a_j(x)$ is identified the j-th neuron output when x is given as input, while m is the number of observed samples. By means of the KL divergence penalty term incorporated in the objective function, $\hat{\rho_j}$ can be enforced to approximately be equal to another value ρ, called *sparsity parameter* – which typically is a small number.

The KL divergence penalty term can be defined proportional to:

$$\sum_j \left[\rho \log\frac{\rho}{\hat{\rho_j}} + (1-\rho)\log\frac{(1-\rho)}{(1-\hat{\rho_j})}\right] \tag{4.1}$$

which is minimized – and equal to 0 – whenever $\hat{\rho_j} = \rho$. In figure 4.2 is shown the trend of KL divergence term when $\rho = 0.2$.

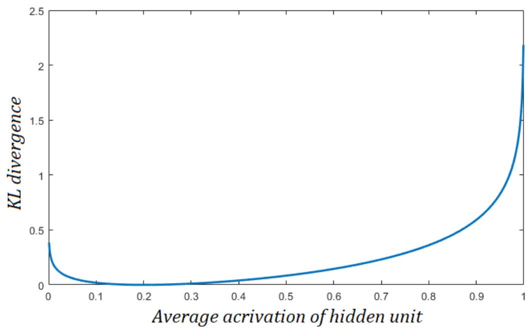


Figure 4.2: KL divergence plot with $\rho = 0.2$

The definition of 4.1 is based on the concept of KL divergence applied to Bernoulli random variables with average ρ and $\hat{\rho}_j$. Indeed:

$$D_{KL}(\rho||\hat{\rho}_j) = \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{(1-\rho)}{(1-\hat{\rho}_j)}$$

In addition to the sparsity penalty, it was chosen to introduce also a *weight decay* term, also known as *L2 regularization*, which penalizes high-valued coefficient in order to avoid overfitting problems, since it shrinks the model. More precisely, it is a penalty term proportional to the square of the magnitude of the coefficients.

The resulting objective function of the deployed SAE becomes:

$$\mathcal{L}(x) \quad = \quad MSE(x, \hat{x}) + \beta \sum_j D_{KL}(\rho||\hat{\rho_j}) + \omega\psi^2$$

The entire encoding mechanism is schematized in figure 4.3

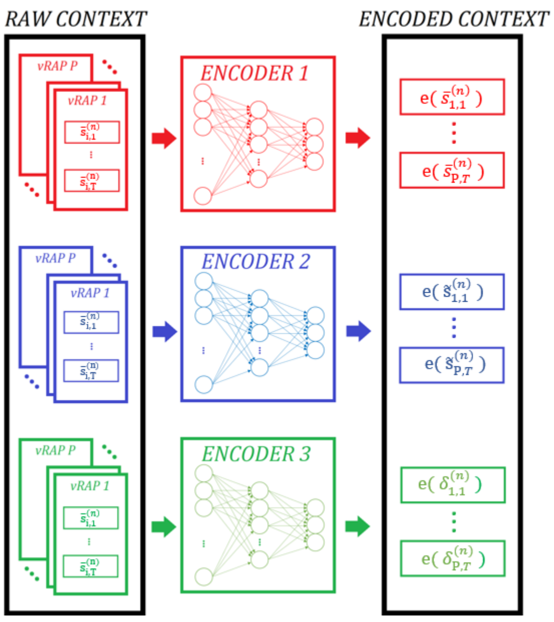


Figure 4.3: Encoder mechanism scheme

Such a scheme shows that each feature of the context information is encoded separately, for each vRAP, and then the results are merged in an unique vector which contains the whole encoded context information.

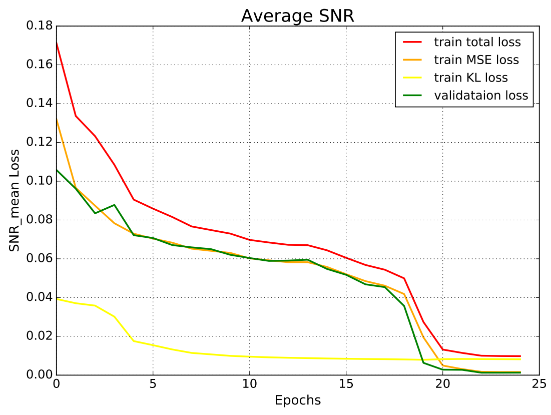
4.2.2 Encoder: hyperparameter tuning and training

All the three encoders have been chosen to have the same structure: the input layer size is obviously equal to the number of samples per stage, namely 200; then there are three hidden layers of sizes $\{100, 20, 4\}$ and finally the output layer has size $D_y = 4$. That means that the input information context of 200 samples is reduced in a 4-dimensional representation. The encoder is then followed by the decoder which has a symmetric structure.

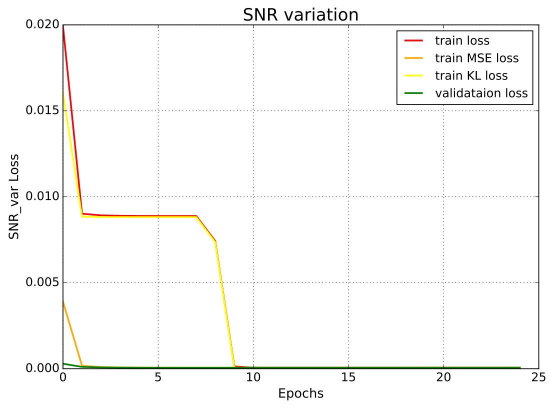
The adopted activation function is the ReLU, in all the layers of the SAE. Such a function is defined as:

$$f(x) = max(0, x)$$

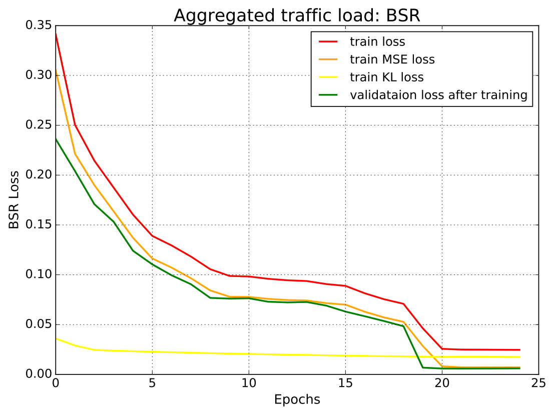
The resulting system is then trained exploiting *PyTorch*, an open source machine learning library. Two-thirds of the dataset obtained as described in 4.1 were used to train the models, exploiting the Adam gradient descend algorithm. Adam optimization algorithm is a valid alternative to classical stochastic gradient descent algorithm that dynamically adapts the learning rate for each network parameter rather than maintaining fixed a single learning rate for all of them. The dataset remainder was used for validation.



(a) Average SNR



(b) SNR variation



(c) Aggregated traffic load

Figure 4.4: Autoencoder learning curves with $lr = 0.001$, $\beta = 0.01$, $\rho = 0.2$, $\omega = 0.01$.

Figure 4.4 reports the obtained learning curves for all three the autoencoders: it has been plotted the curves relating to MSE train loss, sparsity train loss, total train loss – which is the sum of the first twos – and validation loss.

More precisely, the plots have been obtained setting:

- Learning rate $lr = 0.001$.

- Batch-size of 100 stages, i.e. $100 \times 200 = 20000$ samples.
- 25 epochs.
- Sparsity weight $\beta = 0.01$
- Sparsity parameter $\rho = 0.2$
- Weight decay parameter $\omega = 0.01$

No pre-training technique was applied other than data normalization.

Figure 4.4 shows that the curves obtained for the average SNR (4.8a) and the aggregate traffic load (4.4c) are similar. The convergence is obtained after 20 epochs, when also the sparsity penalty term is reduced.

The convergence for SNR variance (4.8b) is instead obtained really quickly: two epochs only would be sufficient to obtain an almost zero MSE loss. This is due to the fact that only a stationary user, therefore whose SNR variance is zero, is considered.

In the following plot series (figure 4.5), instead, it is intuitively evaluated the behaviour of the reconstructed signals, together with the decoded ones. Only the average SNR is considered, since the behaviours of the other autoencoders are comparable.
The first signal, which represents a constant average SNR, is perfectly reconstructed by the decoder.

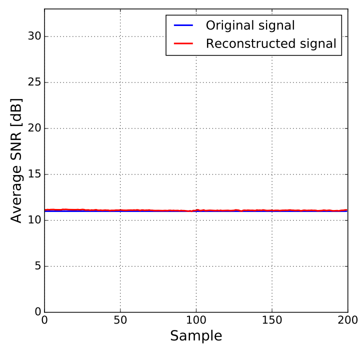
The second one instead presents, during the stage duration, a low variation, caused by the noise added to the rounding quantization of the carried out simulations, which is not captured by the autoencoder: the reconstructed signal is approximately its average value. This behaviour is caused by the limited dataset which was exploited during the training phase: the great majority of the context samples was stationary.

Finally, some signals belonging to the dataset were not perfectly reconstructed, as shown in figure 4.5c. Clearly this loss of information can occur whenever a dimensionality reduction is applied. But apart from some unusual cases, the decoded signal reconstructs the original, so much so that the validation MSE loss is lower than 0.01.

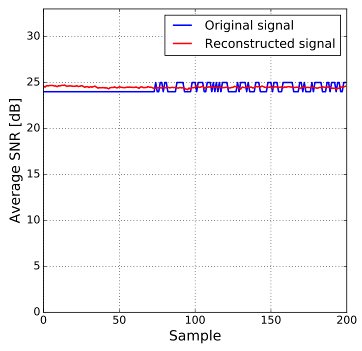
4.3 Radio policy

The encoder is followed by the proper resource manager, the entity which, by understanding the relationship between radio solutions and the corresponding computational load, has to take a valid decision for the following stage, namely it distributes the computational resources among the vRAPs and defines a maximum eligible MCS for them.

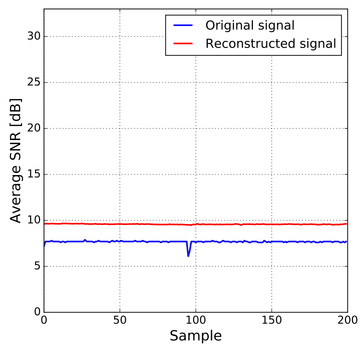
The resource manager is composed by the CPU policy, that defines the criterion based on which distributing the computing capacity, and the Radio policy, which instead outputs, for every vRAP, the corresponding maximum eligible MCS based on the relative allocated CPU. The resource manager has been so decoupled in two entities, one responsible for the computational resources, the other one for the radio resources. Even if, as already outlined in section 2.4, radio and computational resources are strongly correlated, this separation has been possible deploying a *deterministic* Radio policy. In other words, given an information context, the choice made by the Radio policy is known once that the



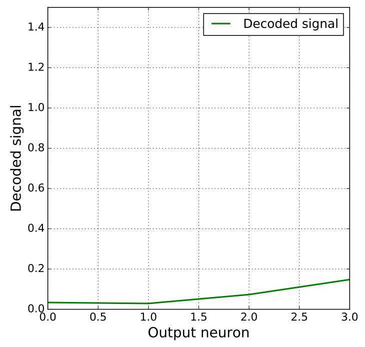
(a) First original and reconstructed signal



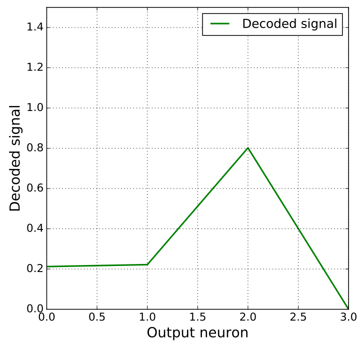
(b) Second original and reconstructed signal



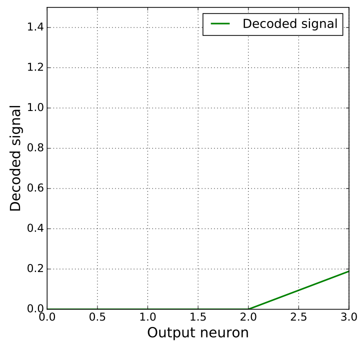
(c) Third original and reconstructed signal



(d) First encoded signal



(e) Second encoded signal



(f) Third encoded signal

Figure 4.5: Examples of comparison between original and reconstructed average SNR signals (top) and corresponding encoded signals (bottom) – [$lr = 0.001$, $\beta = 0.01$, $\rho = 0.2$, $\omega = 0.01$].

CPU allocation is expressed. The correlation between radio and computational resource is entirely taken on by the CPU policy.

In this paragraph it is investigated the design for the Radio policy, even if it is the last acting entity of the entire system. In order to do so, the allocated CPU will be assumed as known. The CPU policy design will instead be described in section 4.4.

4.3.1 Radio policy design

As already mentioned, the Radio policy has the task of expressing, for each vRAP, an upper limit for the selectable MCSs. This step is required in a dynamic resource controller to avoid CPU wastages and performance degradations due to computational outages [7]. The latter occur whenever the carrying out of a certain task – in RAN design case, the decoding of a sub-frame – takes too long due to the limited CPU capacity allocated for such a task. And especially for communication systems this matter is particularly sensitive, since communications protocols present tight time constraints. Avoiding CPU outages is crucial for maximizing the system throughput and consequently minimizing the delays.

And since the computational load increases with the modulation and coding scheme index, as explained in section 2.4, it is surely needed a criterion according to which limiting the eligible MCSs whenever the allocated CPU is bounded. If the latter is instead plenty, then the Radio policy may not impose any constraints.

Once that this upper bound is expressed, the vRAP can select the MCS to be adopted following traditional *computation-unaware MCS selection* procedures, ensuring that the chosen MCS does not exceed the given bound.

Such computation-unaware procedures are not defined in LTE specifications, but they are generally based on the experienced channel quality: the higher the CQI, the larger the MCS selected, since the radio constraints are relaxed. In particular srsLTE provides a first mapping between SNR and CQI values, and a second one between CQI and MCS. More complex mechanisms are based on the BSR, also. This latter may have an impact on the MCS choice since lowly loaded users without great demands on throughput and delay could afford lower MCSs even in good radio conditions.

The Radio policy presented in this thesis work relies on an iterative process in which, given the information context and the computational allocation, the upper bounds for the MCS are tested in descending order until a *binary classifier* estimates that the corresponding *decoding error probability* $\hat{\epsilon}$ is below a certain *threshold* γ. Such an iterative mechanism is showed in figure 4.6.

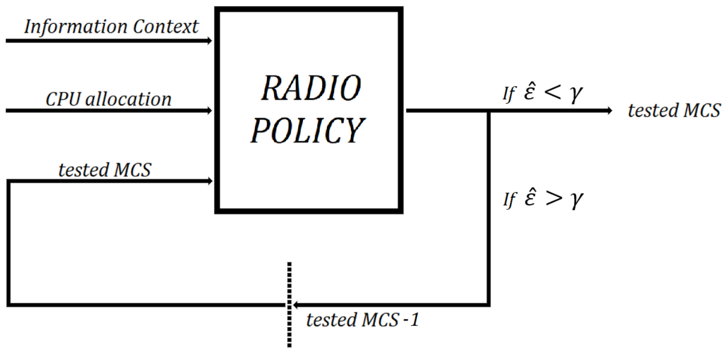


Figure 4.6: Radio policy iterative loop

The core of the Radio policy block is so the classifier.

At large, a classifier is a supervised ML algorithm that has the task of assigning the input data to a class among those available. More precisely, the algorithm does not explicitly express the class, but a confidence level for each of the classes.

In case of multi-class classification problems, namely when there are more than 2 possible classes, a neuron at the NN output layer is attributed to each of the classes. Each neurons outputs the confidence level, namely a probability, with which the input data is estimated to belong to the corresponding class.

Instead, in case of binary classification problems, namely when only 2 classes are defined, it is enough to have just one output neuron associated to one of the two classes, since the confidence level of the other one can be computed by difference.

The classifier of the presented Radio policy is a binary classifier which has been deployed exploiting a simple NN that takes as input the encoded information context and the CPU allocation – given by the CPU policy –, added to the tested MCS upper bound. Then the NN outputs the probability that the estimated decoding error probability associated to the input data and the tested MCS upper bound is *above* the γ threshold. If this last is considerable, the tested MCS upper limit is too large and has to be reduced.

4.3.2 Radio policy: hyperparameter tuning and training

The proposed system envisages the deployment of one Radio policy per vRAP. Each Radio policy resorts to a NN binary classifier with the following structure:

- The input layer has a number of neurons equal to the encoded context size $dim(y)$[3] plus two, since also the CPU allocation and the tested maximum MCS are given as input.
- Eleven hidden layers of sizes $\{5, 8, 20, 30, 40, 40, 40, 40, 30, 20, 5\}$ activated by ReLU functions.
- One-neuron output layer which yields a confidence level – that has to be comprised in $[0; 1]$ by definition – with which the input data are estimated to be associated with a decoding error probability higher than the threshold. In order to respect the constraint on the range of the output value, the output neuron is activated with the sigmoid function[4].

The chosen threshold for the decoding error probability is $\gamma = 0.01$. Therefore a relative decoding error of up to 1% is permissible.

The training of the resulting model has been driven by means of the dataset derived from the conducted simulations described in 4.1 to which it was added one feature containing the class label. More precisely, each sample that showed a decoding error probability greater than γ was labelled as class 1. The other ones as class 0.

The adopted loss criterion is the binary cross-entropy (BCE), defined as:

$$BCE(x, y) = -\frac{1}{N}\sum_{i=1}^{N}\Big[y_i \cdot \log\Big(p(x_i)\Big) + (1 - y_i) \cdot \log\Big(1 - p(x_i)\Big)\Big]$$

where x_i is the i-th of the N samples given as input, $p(x_i)$ the corresponding output of the NN and y_i the corresponding label. The BCE loss function is obviously minimized when the predictions x_i are equal to the labels y_i and it is widely used for binary classifiers.

The optimizer is again the Adam one.

[3]The encoded context size is equal to the number of vRAPs (P), times the number of features belonging to the context information (3), times the encoder output layer size (4). Namely, the encoded context size is $dim(y) = 12P$

[4]The sigmoid activation function is defined as: $\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$. Its domain is the set of real numbers, while its range is comprised in $(0; 1)$

Finally, the tested maximum MCS is selected when the Radio policy predicts that the decoding error probability corresponding to the inputs belongs to class 1 with a probability smaller than 0.5, namely the Radio policy is more confident classifying the input's estimated decoding error probability below the threshold.

Again, two thirds of the dataset was used to conduct the learning phase of the model, while the remainder was used for test and validation. The algorithm was deployed exploiting the PyTorch library. Figure 4.7 shows the learning curves obtained with the following hyperparameters setting:

- Learning rate $lr = 0.001$.
- Batch-size of 100 stages, i.e. $100 \times 4 = 400$ samples – remembering that the decoded context has 4 dimensions.
- 20 epochs.
- Decoding error probability threshold $\gamma = 0.01$

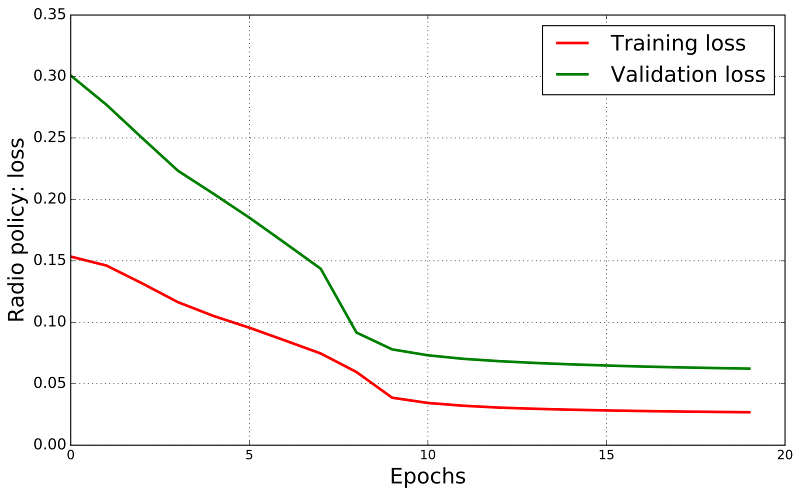


Figure 4.7: Radio policy: learning curves

4.4 CPU policy

The CPU policy manages all the available CPU resources C for the BBU pool. Its purpose is to allocate, for every stage n, a certain portion $c_i^{(n)}$ of the total capacity to each vRAP i. If normalization with respect to the total capacity C is applied, then it must hold:

$$\sum_{i=0}^{P} c_i^{(n)} = 1 \tag{4.2}$$

It is worth to note that the the expression 4.2 involves $P + 1$ terms, since also the unallocated CPU capacity $c_0^{(n)}$ is taken into account.

4.4.1 CPU policy design

The described scenario naturally falls into a class of RL problems called *contextual-bandit* (CB), which is a particular version of one of the most classic RL paradigm: the *multi-armed bandit problem*. The latter was born from the idea of a gambler who has to decide with which slot machine of those present to play, how many plays has to make before changing the slot machine and in what order to switch to the others. But the general idea of multi-armed bandit problems is the presence of a limited and fixed amount of resources (in the previous example, the tokens) that must be allocated among competing choices (in the previous example, the slot machines) in order to maximize the expected gain.

CB problems instead also involve, at each step, an observation that the agent can exploit, together with its past experience, in order to distribute the available resources.

Hence the CPU policy can be formulated as a CB problem.

The accomplishment of Radio policy design comes with several challenges:

- At first, as for every RL problem, a reward signal has to be defined.
- As it has already been outlined, CPU and Radio resources are strictly correlated. As a consequence, a resource manager must jointly consider both in order to achieve optimum performance.
- The action space is continuous. Indeed, the CPU portions $c_i^{(n)}$ to be allocated to each vRAP are real valued. Not all RL algorithms efficiently and sufficiently well work with continuous action spaces.

The reward signal design has to translate the behaviour of the system: it has to penalize or award a prize to the controller if it causes the system to, respectively, malfunction or functioning. Therefore, at the basis of the definition of the reward, there is the ideal behaviour of the system.

At first, the system must respect the QoS requirements of the connections. The latter can be generally expressed in terms of several parameters, first among all throughput and latency. However, it has been chosen to adopt as QoS metric a third parameter, closely linked to the first two, which is the target buffer occupancy Q_i. The higher the throughput required by the i-th vRAP $-$ or, equally, the lower the latency required $-$, the lower will be the relative target buffer occupancy. This also introduces the possibility to differentiate the supported vRAPs based on their QoS requirements. The system satisfies the requirements of the i-th vRAP when its actual buffer occupancy q_{i,x_i,a_i}, after that the context x_i has been observed and action a_i taken, is lower than the target one. So, it can be introduced the quantity:

$$J_i(x_i, a_i) \coloneqq P\Big[q_{i,x_i,a_i} < Q_i\Big]$$

In order to keep J_i high, the agent must ensure that the average decoding error probability ϵ_i of the i-th is maintained bounded. Simultaneously, the agent should limit the operation costs in terms of CPU usage.

The deriving definition of the reward associated to the state-action pair (x, a) is the following:

$$r(x, a) \coloneqq \sum_{i=1}^{P} \Big[J_i(x_i, a_i) - M\epsilon_i - \lambda c_i \Big] \tag{4.3}$$

The definition of M and λ represents the trade-off between computing and radio resources. M is set to a large value in order to keep the decoding error probability low, while λ is set to a small value, since the minimization of the exploited computational capacity must only take place as long as the system reaches good performance. That means that the reduction of operation costs has a secondary priority.

It is worth to note that the reward signal clearly depends on both CPU and radio choices: it translates the behaviour of the entire resource manager. This leads to the second issue previously introduced and causes that the designing of the CPU policy must take into account both radio and CPU resources, namely it must be aware of the Radio policy choices and of their effects. But since the Radio policy has been designed in section 4.3 as a deterministic classifier, then it can be considered by the CPU policy as being part of the environment. This does not affect the optimality of the controller: the optimization of the computational resources management entails that for the radio resources, also.

The reward signal can be hence redefined as:

$$r(y, c) \coloneqq \sum_{i=1}^{P} \Big[J_i(y_i, c_i) - M\epsilon_i - \lambda c_i \Big]$$

The last expression is equal to 4.3, in which however the information context x has been substituted with the encoded one, y, while the action a has been replaced with the CPU allocation, c. This last passage reiterates that the system's optimality entirely depends on the action chosen by the CPU policy, once the radio policy is deterministic.

To recap, the CPU policy is an entity that must optimally distribute among the directed vRAPs the total computational capacity subjected to the constraint 4.2, efficiently coping with real valued actions, in order to maximize the expected reward:

$$R \coloneqq \mathbb{E}_y \left[r(y, c) \right] \tag{4.4}$$

Being the target policy deterministic, the expectation 4.4 only depends on the environment. Hence, the CPU policy can be learnt off-policy.

A *deep deterministic policy gradient* (DDPG) algorithm, based on a model-free actor-critic approach [15], is then resorted. The latter, contrarily to "*Deep Q Network*" (DQN) algorithms, which are an extension of Q-learning solutions combined with a deep NN function approximator that estimates the action-value function, can be applied to continuous action spaces. DQN algorithms indeed, even if operate well with high-dimensional state spaces, can not cope with real valued actions as they attempt to find, at each time step, the best action, which requires to inspect infinite possibilities.

Before introducing the proposed solution, it is worth to have a brief overview about DQN. The latter has established the exploitation of non-linear approximators for action-value functions, which have been historically avoided in RL due to the lack of theoretically and practical guarantees about their convergence and stability. This issue was solved in DQN algorithms by introducing two innovations:

- At first, since the learning of the approximator assumes its training samples to be independently and identically distributed, a *replay buffer* is needed. Training samples, indeed, can not derive *online* from the agent's exploration policy, as consecutive states are correlated. The replay buffer is a finite cache memory in which experienced trajectories are stored and randomly sampled.

- Implementing directly the Q-learning rule (3.24) makes the system to diverge, since the action-value function approximator is involved both on the expected and on the estimated values. To solve this problem, two approximators are implemented in place of one. The first one − which is named *target* − provides the estimates for action-value function updates. The other one instead encompasses the updates. Only after the update, the two are synchronized with each other. In this way the problem of a moving target is avoided.
  The synchronization between approximators takes place via *soft-updates*. By identifying the parameters of the target network with Θ' and the ones of the corresponding learned network with Θ, the soft-update is defined as:
  $$\Theta \longleftarrow \tau\theta + (1-\tau)\Theta' \tag{4.5}$$
  where it is set $\tau \ll 1$ in order to make the target network change slowly and the learning stable.

In DDPG the main features of DQN are held and, in order to deal with real valued action, also the agent is approximated with a target NN $\mu(s|\theta^{\mu})$ with parameters θ^{μ} − and the corresponding learned network obviously, $\mu'(s|\theta^{\mu'})$ with parameters $\theta^{\mu'}$. The critic network $Q(s,a|\theta^{Q})$ has instead parameters θ^{Q}.

The critic is optimized, as in DQN, by minimizing the loss:

$$\mathcal{L}(\theta^{Q}) = \mathbb{E}[(Q(s_t,a_t|\theta^{Q}) - y_t)^2] \tag{4.6}$$

where

$$y_t = r(s_t,a_t) + \gamma Q(s_{t+1},\mu(s_{t+1})|\theta^{Q})$$

The latter expressions are nothing but the standard Q-learning update.

The actor instead has to rely on *Deterministic Policy Gradient Theorem* [16], which is obtaining by maximizing the mean estimates given by the critic:

$$\nabla_{\theta} J(\mu_{\theta}) = \mathbb{E}\left[\nabla_{\theta}\mu_{\theta}(s)\, \nabla_a Q^{\mu}(s,a)\big|_{a=\mu\theta(s)}\right] \tag{4.7}$$

where J identifies the cumulative reward.

As in off-policy algorithms, an independent exploration method can be deployed. The exploration method directly contributes to the choice made by the actor, by adding to the latter a component of noise as follows:

$$\mu'(s_t) = \mu(s_t|\theta_t^\mu) + \mathcal{N}$$

In [15] it has been introduced the Ornstein-Uhlenbeck process in order to generate temporally correlated noise samples $\mathcal{N}$.

The resulting algorithm can be schematized as follows [15]:

Algorithm 1: DDPG algorithm

```
Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$
  and $\theta^\mu$
Initialize target network Q' and $\mu'$ with weights $\theta^{Q'} \longleftarrow \theta^Q$, $\theta^{\mu'} \longleftarrow \theta^\mu$
Initialize replay buffer $R$
for  episode = 1,M do
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial observation state $s_1$
    for  t = 1,T do
        Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and
          exploration noise
        Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{+1t}$
        Store transition ($s_t$; $a_t$; $r_t$; $s_{t+1}$ in R
        Sample a random minibatch of $N$ transitions ($s_i$; $a_i$; $r_i$; $s_{i+1}$ from R
        Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
        Update critic by minimizing the loss: 4.6
        Update the actor policy using the sampled policy gradient: 4.7
        Update the target networks by soft-update rule: 4.5
    end
end
```

4.4.2 CPU policy: hyperparameter tuning and training

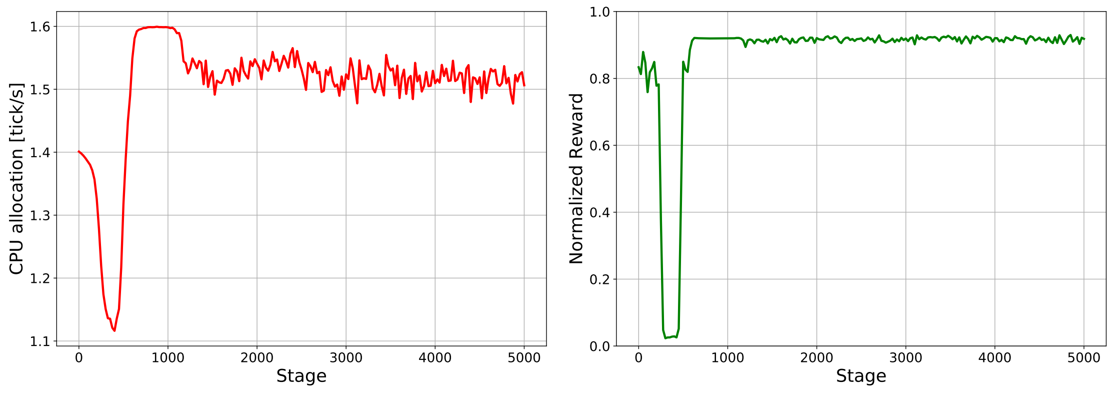
Algorithm 1 needs to be slightly revisited and readjusted to be implemented by the CPU policy. This is due to the fact that the time domain was first divided into stages, then into slots. But while the CPU policy acts at stage's timescale, the reward is received once per slot. Therefore, the way in which the CPU policy operates is the following: at the beginning of each stage, the CPU policy, based on the context measured during the previous stage, makes a decision regarding the next one. Then for the entire stage duration, it receives rewards. The latter are then averaged: the stage's average reward is the quantity that drives the definition of state-action value functions.

Also the expression 4.2 has to be adapted: the normalization of the CPU values was based on the range of simulated CPU allocations, which is $[1.1, 1.6]$.

Both the actor and the critic networks have equal hidden layers with 20,40,80,20,10 hidden neurons. The latter are activated by the ReLu function. The actor and critic NNs obviously differ each other for the input and output layers. The actor takes as input the encoded context only, which has dimension $dim(y)$, while it outputs a CPU allocation for all the active vRAPs plus the saved capacity. The output layer is activated by the soft-max function in order to respect the constraint 4.2. The critic's input instead is formed by the encoded context and the allocated CPU capacity $-\ dim(y)+1\ -$ while it outputs an action-value function estimate. Its output layer is consequently formed by just one neuron.

The controller has been deployed exploiting PyTorch and trained relying on Adam optimizer. These are the general implemented settings:

- Neural networks learning rate, $lr = 0.0001$.
- Reward discount factor, $\gamma = 0.99$.
- Soft-update weight, $\tau = 0.01$.
- Buffer capacity of 1500 trajectories.
- Minibatch size of 300 trajectories.
- Reward: Decoding error probability weight, $M = 2$.
- Reward: CPU time allocation weight, $\lambda = 0.25$.
- Noise samples taken from an Ornstein–Uhlenbeck process with $mu = 0$, $\theta = 0.001$, $\sigma = 0.001$.



(a) CPU capacity allocation temporal evolution (b) Achieved reward temporal evolution

Figure 4.8: CPU Policy learning curves.

Figure 4.8 shows the performance obtained by the entire system $-$ which instead is represented in figure 4.9$-$ with a target buffer occupancy of $10000\,Bytes$. In both sub-figures it is possible to notice a first phase in which the controller is exploring and therefore performs actions that result in very low rewards. Instead, it follows a phase in which the controller has learned which are the best allocations to perform and in which states, so that the obtained reward is almost maximized. For visualization purposes, the reward has been normalized in the range $[0; 1]$, where 0 means that the QoS metric is never respected,

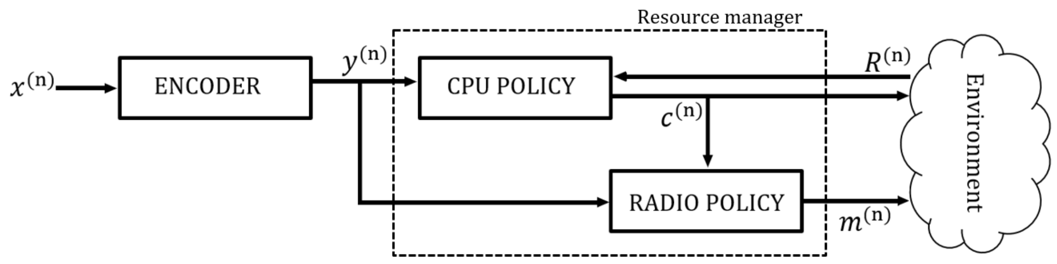


Figure 4.9: Dynamic resource controller − scheme

100% decoding error probability, and the allocation of the entire CPU capacity; while 1 represents the opposite condition. The obtained average reward is, excluding the learning phase, 0.92.

Chapter 5

Conclusions and Future Work

In the previous chapters we have analyzed in detail the proposed design for a dynamic controller of computational and radio resources. This is a very promising system that potentially meets the stringent needs of telephone operators for a virtual access network that is able to meet the demands of the heterogeneous spectrum of users.

Clearly, during the thesis work it was possible to approach the design of the system by analyzing its performance only in a simplified context. The relative achieved results were definitely satisfactory and optimistic: the reward obtained from the system stabilizes at a sensibly high value, while the CPU allocation is dynamically controlled. This indicates that the whole system has managed to understand the complex relationship that exists between computational and radio resources by varying the environment conditions, namely the context. So the three entities forming the dynamic resource controller optimally operate and cooperate: the encoder satisfactorily translates the information context, the CPU policy efficiently manages the computational resources and the Radio policy accurately limits the radio resources.

The next step concerns the analysis of the operation of the same system in much more complicated conditions, which first involve the presence of multiple users connected to the virtual access point. Users who are no longer stationary, but that experience variable radio conditions. Furthermore, multiple access points can be considered. And everything can and actually must be made even more complex if users are diversified according to their QoS requests. And considering more complicated states will likely also require re-optimization of the system parameters.

Another fundamental step for the evaluation of the controller concerns an analysis of the computational costs that it entails, although it is based on neural networks, therefore on very efficient elements from this point of view. This step is necessary to be able to compare this solution with others.

On this regard arises also the optimization of the trade-off between computational costs and performance, which can be translated into the definition of stage and monitoring slot durations.

Bibliography

[1] Cisco Systems (2020). *Cisco visual networking index: global mobile data traffic forecast update, 2018-2023* [White Paper] Retrieved on September 15, 2020 from Cisco: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[2] Jyotsna Agrawal;Rakesh Patel; P.Mor; P.Dubey and J.M.keller. *"Evolution of Mobile Communication Network: from 1G to 4G"* International Journal of Multidisciplinary and Current Research, Vol. 3, pp. 1100-1103.

[3] document ITU-R M.[IMT-2020.TECH PERFREQ], Oct. 2016. *"Minimum Requirements Related to Technical Performance for IMT-2020 Radio Interface(s)".*

[4] China Mobile Research Institute, *"C-RAN. The Road Towards Green RAN"*, Whitepaper v. 2.6, Sep. 2013.

[5] A. Checko; H. Christiansen; and M. S. Berger; *"Evaluation of energy and cost savings in mobile Cloud RAN"* in Proc. OPNETWORK Conf., Washington DC, USA, 2013, p. 8.

[6] Yonghua Lin; Ling Shao; Zhenbo Zhu; Qing Wang; Ravie K. Sabhikhi. *"Wireless network cloud: Architecture and system requirements"*, IBM J. Res. Dev. 54 (1)(2010), 4–1.

[7] M. C. Valenti; S. Talarico; and P. Rost. *"The Role of Computational Outage in Dense Cloud-Based Centralized Radio Access Networks*, in IEEE Global Conference on Communications, (Austin (TX), USA), December 2014

[8] D. Bega; A. Banchs; M. Gramaglia; X. Costa-Pérez and P. Rost. *"CARES: Computation-aware Scheduling in Virtualized Radio Access Networks"* in IEEE Transactions on Wireless Communications, vol. 17, no. 12, pp. 7993-8006, Dec. 2018, doi: 10.1109/TWC.2018.2873324.

[9] P. Rost; S. Talarico and M. C. Valenti. *"The Complexity–Rate Tradeoff of Centralized Radio Access Networks"*, in IEEE Transactions on Wireless Communications, vol. 14, no. 11, pp. 6164-6176, Nov. 2015, doi: 10.1109/TWC.2015.2449321.

[10] Ethem Alpaydın, *“Introduction to Machine Learning”*, The MIT Press, Cambridge, Massachusetts, London, England, 2010, p. 3.

[11] David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams, *“Learning representations by back-propagating errors”*, Nature 323, 533–536 (1986).

[12] van Otterlo, M., Wiering, M., (2012), *“Reinforcement learning and Markov Decision Processes”*. In: Wiering M., van Otterlo M. (eds) Reinforcement Learning. Adaptation, Learning, and Optimization, vol 12. Springer, Berlin, Heidelberg.

[13] Sutton, R. S., Barto, A. G., (2018), *“Reinforcement learning: an introduction*, The MIT Press.

[14] Hinton, G. E.; Salakhutdinov, R.R. (2006-07-28). *"Reducing the Dimensionality of Data with Neural Networks"*. Science. 313 (5786): 504–507.

[15] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. *"Continuous control with deep reinforcement learning"*. In Proceedings of the 2016 International Conference on Learning Representations (ICLR 2016), San Juan, Puerto Rico, May 2016.

[16] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, *“Deterministic policy gradient algorithms”*, 2014.