University of Khartoum
Faculty of Engineering
Department of Electrical and Electronic Engineering

Autonomous Cellular Network Management

A thesis submitted in partial fulfillment of the requirements of the B.Sc. degree in Electrical and Electronic Engineering

Submitted by

Ahmed Mohammed Abdelkareem Alhassan 154012

Merghani Ismail Braima Abdelsamad 154102

Supervised by

Dr. Anas Showk

May 2022

Dedications

To our families for their unconditional love and continuous support, and to the people of Sudan.

Declarations

We, **Ahmed Mohammed Abdelkareem Alhassan**, **Merghani Ismail Braima Abdelsamad**, declare that this thesis titled, "**Autonomous Cellular Network Management**", and the work presented in it are our own. We confirm that:

- This work was done wholly or mainly while in candidature for a B.Sc degree at the University of Khartoum.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where We have consulted the published work of others, this is always clearly attributed.
- Where We have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- We have acknowledged all main sources of help.

Ahmed Mohammed Abdelkareem Alhassan
154012
Merghani Ismail Braima Abdelsamad
154102
September 16, 2025

Acknowledgments

This research would not have been possible without the support of many people. Many thanks to our adviser, Dr. Anas Showk and also Dr. Mohammed Abbas who reviewed our numerous drafts, provided guidance through out the research process, and helped make some sense of the confusion. We would also like to extended our gratitude to Dr. Mohamed Hamid, Muhammed Saeed, Amr Alamin, Moayad Hassan, Mohammed Nagdi, and Baraa Adil for their technical support and valuable time.

المستخلص

يعيش السودان حالياً في ظروف لا يمكن التنبؤ بها ، بما في ذلك التضخم المفرط وضعف البنية التحتية والاقتصاد المنهار. في ظل هذه الظروف ، أعلنت وزارة الطاقة الحالية أن نظام إمداد الطاقة يعاني من ندرة في المكونات البديلة ، وأن أزمة في قطاع الكهرباء ستؤدي إلى نتيجة مباشرة لهذه القيود ، مما يؤدي إلى عدم الاستعداد الكافي للطلب الوارد. ونتيجة لذلك ، تم سن التشريع الأخير لرفع تكاليف الطاقة والبدء في برامج فصل الأحمال. تعرضت شركات الاتصالات لضربة كبيرة نتيجة الأزمة الكهربائية المستمرة ، والتي تجبرها على تقديم الخدمة في ظل ظروف غير موثوقة للغاية ، مثل تشغيل المحطات الأساسية في الأيام التي تنقطع فيها الكهرباء لمدة 8 ساعات في المتوسط ، مع ضمانات قليلة أو معدومة. حول جودة الخدمة المقدمة للعملاء ، وخاصة مع الزيادة المستمرة في عدد السكان في المدن الكبرى مثل الخرطوم ، فمن الصعب تقديم الخدمة. الهدف من هذا المشروع هو إنشاء برنامج قابل للتطوير يمكنه إعادة إنشاء الحالة غير المستقرة الحالية للبلد إلى حد كبير ثم تطوير نظام مستقل فوقه يمكنه تقديم اقتراحات صحيحة وإدارة شبكة المحطات القاعدية بالكامل. كان أداء النماذج المدربة أفضل من نموذج التشغيل البسيط. حققت هذه النماذج كفاءة عالية من خلال تشغيل عدد كافٍ من المحطات الأساسية لتحقيق جودة عالية للخدمة مع تقليل التداخل بين المحطات القاعدية.

Abstract

Sudan is currently under unpredictable circumstances, which include hyperinflation, weak infrastructure, and a collapsing economy. In these conditions, the current Ministry of Energy has declared that the power supply system is experiencing a scarcity of replacement components, and that a crisis in the electric sector would result as a direct product of these limitations, resulting in insufficient preparedness for incoming demand. As a result, recent legislation to raise power costs and commence load-shedding programs was enacted. Telecommunication companies have been dealt a major blow as a result of the ongoing electric crisis, which forces them to provide service under very unreliable conditions, such as operating base stations on days when electricity is out for an average of 8 hours, little to no guarantees can be made about the quality of service provided to customers, and especially with the ever-increasing population of residents in major cities like Khartoum, it is a difficult task to provide service. The goal of this project is to create a scalable software that can recreate to a considerable part the country's current unstable condition and then develop an autonomous system on top of it that can make correct suggestions and manage the whole network of base stations. Models trained performed better than a simple operation model. Those models achieved high efficiency by operating enough base stations to achieve high Quality of Service while minimizing the interference among base stations.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

A2C	**A**dvantage **A**ctor-**C**ritic
ANN	**A**rtificial **N**eural **N**etwork
BS	**B**ase **S**tation
DDPG	**D**eep **D**eterminstic **P**olicy **G**radient
DQN	**D**eep **Q**-**N**etwork
eNB	**E**volved **N**ode **B**
HER	**H**indsight **E**xperience **R**eplay
HeNB	**H**ome **eNB**
IL	**I**mitation **L**earning
LTE	**L**ong-**T**erm **E**volution
MC	**M**onte **C**arlo
MDP	**M**arkov **D**ecision **P**rocess
MME/GW	**M**obility **M**anagement **E**ntity/**G**ate**w**ay
MS	**M**obile **S**tation
PG	**P**olicy **G**radient
PPO	**P**roximal **P**olicy **O**ptimization
RAN	**R**adio **A**ccess **N**etwork
RL	**R**einforcement **L**earning
SAC	**S**oft **A**ctor-**C**ritic
TD	**T**emporal **D**ifference
TD3	**T**win **D**eployed **DD**PG
UE	**U**ser **E**quipment

List of Symbols

S_t	state at time t
A_t	action at time t
R_t	reward at time t
γ	discount rate (where $0 \leq \gamma \leq 1$)
G_t	discount return at time t($\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$)
S	set of all $non-terminal$ states
S^+	set of all states(including terminal States)
A	set of all actions
$A(s)$	set of all actions available in state s
R	set of all rewards
$P(s^{'}, r\|s, a)$	probability of next state $S^{'}$ and reward r, given current state s and current action a
π	policy if deterministic : $\pi \in A(s)$ for all $s \in S$. if stochastic: $\pi(a\|s) = P(A_t = a\|S_t = s)$ for all $s \in S$ and $a \in A$
v_π	$state-value$ function for policy $\pi(v_\pi = E[G_t\|S_t = s]$, for all $s \in S)$
q_π	$action-value$ function for policy $\pi(q_\pi = E[G_t\|S_t = s, A_t = a]$, for all $s \in S$ and $a \in A(s))$
v^*	optimal $value-function(v^*(s) = max_\pi v_\pi(s)$ for all $s \in S)$
q^*	optimal $action-value$ function $q_\pi(s, a) = max_\pi q_\pi(s, a)$ for $s \in S$ and $a \in A(s)$

Chapter One

Introduction

1.1 Preface

In Sudan Communication technologies have progressed dramatically in both wired and wireless contexts in past few years, providing a broad range of applications and services for hundreds of millions of network users. For instance, Internet has been one of the most notable breakthroughs in the communication field, impacting every part of our lives. Consequently, Internet access and connectivity are now regarded an essential service, similar to water and electricity which bears witness to its importance in modern life. However, in Sudan, the country is dealing with unstable circumstances such as hyperinflation, insufficient infrastructure, and a plunging economy. In these conditions, the current Ministry of Energy has declared that the power supply system is experiencing a scarcity of replacement components, and that a crisis in the electric sector would result as a result of these limitations, resulting in insufficient preparedness for incoming demand. As a result, recent legislation was passed to begin a load-shedding scheme and raise electricity bills. Telecommunication companies had a major blow because of the electric crisis that is still happening which forces them into providing service at a very unreliable circumstances that may include operating base stations at days which electricity may go down for 8 consecutive hours, little to no promises can be made about the quality of service provided to the customers and specially during the ever-growing numbers of the residents in major cities like Khartoum, it is very hard task to adapt to those numbers by managing the whole base station systems manually, so an autonomous system that may recommend how to operate this large system or at best manage the system while reducing human interference would increase the quality of service of the network while optimizing the power usage. With that all the solution of this problem is further optimization of the operation settings which can be made via Autonomous management.

Autonomic network management is a promising strategy for reducing the cost and complexity of network infrastructure management. Autonomic computing intends to

develop self-managing computing systems that govern themselves in the face of high-level administrative requirements. As a developing approach for administering complex networks, both autonomic computing project and autonomic network management (ANM) have been proposed[3].

The comprehensive performance of wireless networks is controlled by how efficiently and proactively resources such as time frames, bandwidths, and orthogonal codes are managed, as well as how well dynamic traffic loads and wireless channel oscillations are fully integrated into network design to deliver connectivity among users with variable Quality-of-Service requirements[4][5].

Optimization has been a valuable tool in various machine learning algorithms and models, since it helps machine learning mechanisms to perform in an optimized way. Given a set of restrictions, optimization attempts to find the optimal element or value that fits an objective function. This value causes the system to work optimally[6].

Traditionally, resource allocation problems are resolved using optimization algorithms that take into account subscribers' real-time QoS requirements. Resource allocation issues are not convex, therefore conventional methods provide solutions that are not globally optimum.More efficient solutions are required for real life implementation. In some circumstances where the allocation of resources problem is not well-defined mathematically we may be unable to articulate the optimization problem. It stimulates the development of novel resource allocation techniques. Data-driven ML-based resource allocation solutions would be effective in such framework and more adaptive to the dynamic nature of networks[4].

ML techniques can be employed when no prior knowledge of the system, network, users, or parameters is provided, and predictions and control decisions must be made. Reinforcement Learning (RL) is one such technique that uses a trial-and-error approach to learn optimum control actions by monitoring unknown parameters and system behavior over time[7].

1.2 Problem statement

The main idea behind this thesis is conjuring a proof-of-concept for autonomous operation for RAN (Radio Access Network) even in an unreliable environment. As the situation in the country is unforgiving, several plans were conducted like load shedding to decrease the severity of the situation. Load shedding has an impact on RAN functioning in many ways, since there are so many factors that can be adjusted to make different sorts of modifications to the RAN, manual methods of operating the RAN requires good intuition and prior experience. And due to limitations of conducting experiments physically, We propose a standardized Open-AI Gym environment to simulate systems of RAN and a deep reinforcement learning model that is capable of

using all parameters available to get the best operation possible that maintains economic feasibility.

The goal of this thesis is to implement a simulator capable of replicating the unreliable conditions (e.g. load-shedding) which a system of RAN goes through in Sudan and teach upon it an intelligent reinforcement learning agent to operate this complex system, using different deep reinforcement learning algorithms to solve the task of operating the RAN system with the least power possible while maintaining QoS (Quality of Service) then benchmark the results of each algorithm using a simple baseline model, and conclude why each algorithm succeeded or failed to solve the task.

1.2.1 Objective

- To perform literature survey of previous methods used in the field of operating RAN (Radio Access Network) systems using deep reinforcement learning.
- To implement a standardized simulator that constitutes the ability of using various types of agents, and simulate various scenarios in a RAN.
- To train different deep reinforcement learning algorithms to operate RAN (Radio Access Network) systems using OpenAI Gym-based environment.
- To benchmark the trained models against a simple model and conclude why each algorithm succeeded or failed to solve the task, achieving this would open up the possibilities of agent to learn to interact with the environment without prior knowledge.

1.3 Methodology

In this work a standardized environment is implemented and then used to benchmark several deep reinforcement learning methods such as policy gradient methods and value-based methods for operating RAN (Radio Access Network) systems in OpenAI Gym simulator. Two deep reinforcement learning algorithms were used: Advantage Actor-Critic (A2C), proximal policy optimization (PPO).

1.4 Thesis Outlines

The rest of this thesis is structured as follows:

Chapter 2 provides a literature review about the fields under study as well as state-of-art in the field. Chapter 3 provides a theoretical background about the fields under study. Chapter 4 provides the System and Environment Modelling for the thesis. Chapter 5

provides the results of the thesis. Chapter 6 is a discussion on the results, Future work, Limitation and conclusion to the project.

1.5 Contribution

- A novelty in the sense of applying deep reinforcement learning in real-time control of RAN.
- A standardized environment which any type of agent can interact with.
- A proof-of-concept of a real life solution for one of the most costly problems in Sudan.
- to-be-submitted Paper on the environment implemented as it is the only standardized environment written in python.
- to-be-submitted Paper on the use of deep reinforcement learning methods for operating RAN (Radio Access Network) systems.

Chapter Two

Background and Literature Review

2.1 Autonomic Networks

Autonomicity of any system describes the capability of system's self-managing given a set of distinguished goals from administrators. High-level goals define for a system what are its objectives and the system pursuits to accomplish them in the best manner. An autonomic system is able to monitor its own performance and adapt itself accordingly, optimize its use of resources and overcome adverse events. This is what differentiates an autonomic system from its automatic counterpart, which can be affected by external factors such as human error or human error[8].

The act of applying autonomic principles to network management is referred to as autonomic networking. Even when the environment changes, an autonomic network works and fulfills its purpose efficiently by regulating itself without external intervention.

The main properties of self-management are described by IBM as follows[9]:

- Self-configuring: This attribute refers to a computer system's ability to configure and reconfigure itself in line with high-level regulations.
- Self-optimization: The goal of self-optimization is to allow the system to operate efficiently even in unpredictable habitat.
- Self-healing: This feature refers to the ability to detect and correct possible issues in order to keep the system running smoothly.
- Self-protecting: Self-protection refers to a system's ability to defend itself against threats that would prevent it from fulfilling its objectives.

2.1.1 The Conceptual Architecture

An autonomic network, by definition, runs and fulfills its goals efficiently by regulating its own self without external intervention, even when the environment changes. Autonomicity may be attained using a distributed architecture composed of an interacting collection of autonomic managers and managed components. An autonomic manager is a self-managed individual component that contributes to the network's overall self-management. It can sense its internal and external environments and act locally on the controlled components it has been allocated. An autonomic manager collaborates with other pairs to ensure that the whole network meets high-level goals. The autonomic manager achieves the self-management thanks to monitoring the managed resources(s) and other autonomic managers, analyzing the collected data, planning adaptation activities and executing the decided plans on the network. This process forms an autonomic control loop (ACL) around network resources[8][3]. IBM proposed a generic reference model called MAPE-K model (Monitor, Analyze, Plan, Execute and Knowledge)[10]. MAPE-K, a feedback control loop envisioned as a sequence of four computations Monitor-Analyze-Plan-Execute over a knowledge base, is a well-known engineering technique to realize self-adaptation. Computations M, A, P, and E may be performed by many components that work together to adjust the system as needed, i.e., they may be decentralized over numerous MAPE-K loops. These MAPE components can communicate directly or indirectly by exchanging knowledge in the knowledge repository[11].

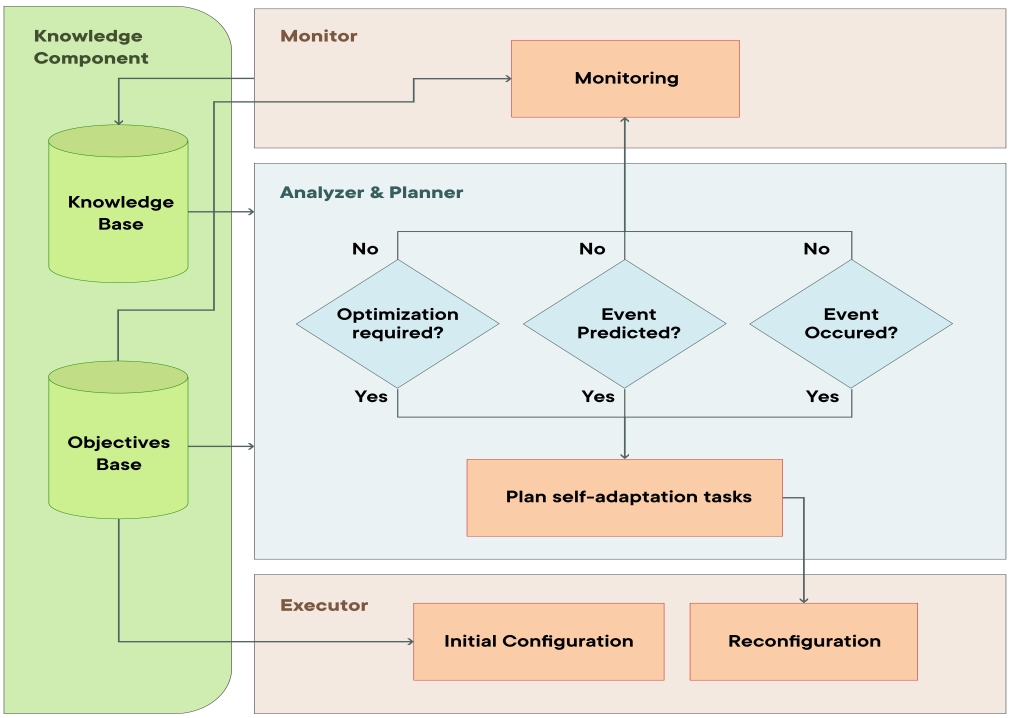


Figure 2.1: Detailed MAPE-K for ACL [11]

The monitor process gets hold of details from managed resources, such as topology information and configuration property settings, by definition. The monitor process collects and filters this information until it identifies a symptom that requires further investigation. The analyze phase does data analysis and reasoning on the symptoms supplied by the monitor process. The analyze process is influenced by knowledge data that has been saved. A change request is sent to the plan process if modifications are necessary. The goal-setting process is structured by the plan process. The planning process develops or chooses a procedure to carry out a planned change in the controlled resources. The planning process can take many different shapes, from a single command to a complicated workflow. The execute process modifies the managed resource's behavior depending on the actions proposed by the plan process. All four processes use the same knowledge base, which contains data such as historical logs, symptoms, and policies. Depending on the outcomes of the procedures, the knowledge base may be updated[12].Sensors and Actuators are software or hardware components that are used to monitor and make changes to the network[8].

2.1.2 The Evaluation Framework

The state of the art to evaluate the performance of a wireless network is to simulate the relevant aspects of the radio access network (RAN) at the system level. The computed results are, for example, system throughput, quality of service (QoS) metrics, and

fairness in terms of cell edge user throughput. Well-defined reference systems and scenarios are established to ensure that results generated by different RAN system simulation tools are comparable. This is the result of substantial standardization bodies consensus work[13].

2.1.3 The Power Model

The BS power model serves as a bridge between component and system levels, allowing for quantification of how energy reductions on individual components improve network energy efficiency. Due to output power, size, and cost limits, the properties of the implemented components are heavily influenced by the BS type. Because of these disparate properties, a power model that is suited to a given BS type is required[13].

2.2 Base Station Power Consumption Breakdown

The figure shows a simplified block diagram of a complete BS that can be generalized to all Base Station types. A Base Station consists of multiple transceivers (TRXs), with each serving one transmit antenna element.

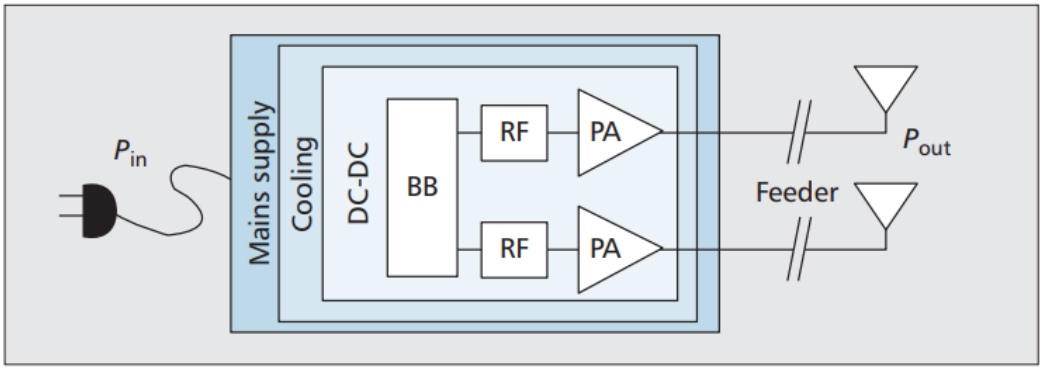


Figure 2.2: Block diagram of a base station transceiver (TRX) [13]

A TRX comprises a power amplifier (PA), a radio frequency (RF) small-signal TRX module, a baseband engine including a receiver (uplink) and transmitter (downlink) section, a DC-DC power supply, an active cooling system, and an AC-DC unit (mains supply) for connection to the electrical power grid. In the following the various TRX parts are analyzed[13].

Antenna interface

The influence of the antenna type on the power efficiency is modeled by a certain amount of losses, including the feeder and antenna bandpass filters. Since macro-BS sites are often situated at different physical locations than the antennas, a feeder loss of

about $\sigma_{feed} = 3dB$ needs to be added. The feeder loss of a macro-BS may be mitigated by introducing a remote radio head (RRH). Feeder losses for smaller BS types are typically negligible.

Power amplifier (PA)

The most efficient PA operating point is close to the maximum output power (near saturation). Nonlinear effects and non-constant envelope signals force the power amplifier to operate in a more linear region (i.e., 6–12 dB below saturation) This prevents adjacent channel interference (ACI) due to nonlinear distortions, and therefore avoids performance degradation at the receiver. The high operating back-off gives rise to poor power efficiency ηPA, and so an increased PA power consumption.

The RF transceiver

For up-link and down link communication, the RF transceiver consists of a receiver and transmitter. The linearity and blocking requirements of the RF module may differ significantly depending on the type of the base station. For macro/micro-BSs, low Intermediate Frequency (IF) or Super-Heterodyne architectures are preferred. Required bandwidth and the allowable signal-to noise-and-distortion ratio (SiNAD), have the most significant impact on the RF energy consumption(PRF)[14].

Base Band Unit (BB)

Performs digital up/down-conversion, including filter modulation/demodulation, digital pre-distortion (only for large BS types), signal detection (synchronization, channel estimation, equalization, compensation of RF non-idealities) and channel coding/decoding. Digital BB also includes the power consumed by the serial link to the backbone network.

Power supply and cooling

Losses incurred by DC-DC power supply, mains supply, and active cooling extent proportionally with the power consumption of the other components, and may be approximated by the loss factors σ_{DC}, σ_{MS}, and σ_{Cool}, respectively. Assuming that the BS power consumption extents proportionally with the number of transceiver chains N_{TRX}, the breakdown of the BS power consumption at maximum load, where $P_{out} = P_{max}$, yields

$$[H]P_{\text{in}} = N_{\text{TRX}} \cdot \frac{\frac{P_{\text{out}}}{\eta_{\text{PA}} \cdot (1-\sigma_{\text{feed}})} + P_{\text{RF}} + P_{\text{BB}}}{(1-\sigma_{\text{DC}})\,(1-\sigma_{\text{MS}})\,(1-\sigma_{\text{cool}})} \tag{2.1}$$

The efficiency is defined by $\eta = P_{out}/P_{in}$, whereas the loss factor σ is defined by $\sigma = 1 - \eta$. Losses due to the antenna interface (other than feeder losses) are not included in the power breakdown. It is seen that the supply power P_{in} scales linearly to the number of TRX chains N_{TRX} (i.e., transmit/receive antennas per site).

2.2.1 Base Station Power Consumption at Variable Load

The BS load, defined as P_{out}/P_{max} in an LTE down-link, is proportionate to the quantity of used resources, which include both data and control signals. In the case of an LTE up-link, the BS load is also affected by power control settings in terms of transmitted spectral power density. The power consumption of the BS as a function of its load demonstrates that it is mostly the PA that rises and falls with the BS load. However, the BS type has a big impact on this scaling. As can be seen in Fig. 3, the relations between relative RF output power P_{out} and BS power consumption P_{in} are nearly linear. A linear approximation of the BS power model appears justified:

$$[H]P_{\text{in}} = \begin{cases} N_{\text{TRX}} \cdot P_0 + \Delta_{\text{p}} P_{\text{out}}, & 0 < P_{\text{out}} \leq P_{\text{max}} \\ N_{\text{TRX}} \cdot P_{\text{sleep}}\,, & P_{\text{out}} = 0 \end{cases} \tag{2.2}$$

where P_0 is the power consumption at the minimum non-zero output power, and Δ_p is the slope of the load-dependent power consumption. The parameters of the linear power model of Eq. 1 for the considered BS types are obtained by least squares curve fitting and are listed in Table 2. Figure 3. Power consumption for various BS types as a function of the load, P_{out}/P_{max}.

2.3 Reinforcement Learning

Reinforcement Learning is a technique for solving sequential decision-making problems with roots in artificial intelligence and computer science[15]. The primary goal of reinforcement learning is to design and train autonomous agents that interact with the environment to learn the optimal behavior. Reinforcement learning methods provide a mathematical framework for experience-driven autonomous learning[16].

Reinforcement learning problems involve learning what to do - how to map situations to actions -so as to maximize a numerical reward signal. In an essential way they are closed-loop problems because the learning system's actions influence its later inputs. Moreover, the learner is not told which actions to take, as in many forms of machine learning, but instead must discover which actions yield the most reward by trying them out. Reinforcement learning involves a closed-loop in an essential way, not having direct instructions as to what actions to take, and where the consequences of actions play out

over extended time periods. The terms supervised learning and unsupervised learning appear to exhaustively classify machine learning paradigms, but they do not. Reinforcement learning is different from supervised learning, the kind of learning studied in most current research in field of machine learning. Supervised learning is learning from a training set of labeled examples provided by a knowledgeable external supervisor. Each example is a description of a situation together with a specification—the label of the correct action the system should take to that situation, which is often called a category. The object of this kind of learning is for the system to extrapolate, or generalize, its responses so that it acts correctly in situations not present in the training set. Reinforcement learning is also different from what machine learning researchers call unsupervised learning, which is typically about finding structure hidden in collections of unlabeled data. The terms supervised learning and unsupervised learning appear to exhaustively classify machine learning paradigms, but they do not. We consider reinforcement learning to be a third machine learning paradigm, alongside of supervised learning and unsupervised learning, and perhaps other paradigms as well. Reinforcement learning is trying to maximize a reward signal instead of trying to find hidden structure. Uncovering structure in an agent's experience can certainly be useful in reinforcement learning, but by itself does not address the reinforcement learning agent's problem of maximizing a rewards signal.

2.3.1 Reinforcement Learning Model

In the standard RL model, an agent interacts with its environment via perceptions (observations) and actions. The agent interacts with the environment in discrete time steps. At each time step the agent receives an input that carries information about the environment at that time step, this input is modelled by the state of the environment. Then the agent applies an action to the environment and receives a real valued reward that tells it how good the action was, this action drives the environment into a new state. The agent guided by RL algorithms learns from the states, actions, and rewards the optimal state-action mapping. The whole process is repeated over and over until the agent learns to act optimally so that it will choose the action that maximizes the long-term sum of rewards[16]. Formally, the RL model consists of:

- A set of environment states, S;
- A set of agent actions, A;
- A set of scalar reinforcement rewards, r;
- A policy π that maps states to actions describing the agent's behavior.

The goal of reinforcement learning is to find a policy π, which will maximize the long-term collection of reinforcements (rewards).

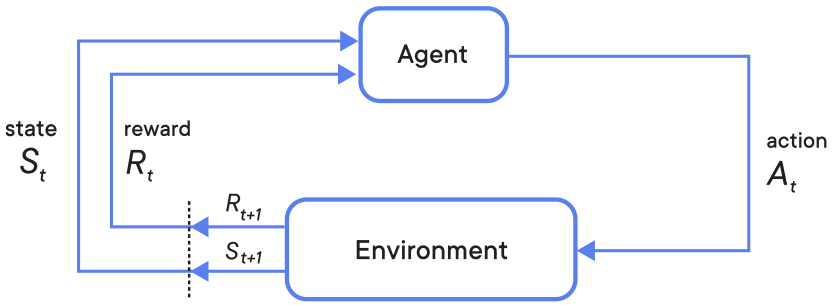


Figure 2.3: The agent-environment abstraction in a reinforcement learning model taken from

2.3.2 Reinforcement Learning Environment

The environment in the reinforcement learning model is assumed to be non-deterministic meaning that taking the same action from the same state on different occasions may lead to different future states or different reward values. However, the environment is assumed to be stationary, i.e., the probabilities of making a state transitions are fixed overtime[16].

2.3.3 The Markov Decision Process

The Markov Decision Process (MDP) is a stochastic model of an environment in which each state within an environment is a consequence of its previous state and the transition from that state to the current one. Each state within an environment is a consequence of its previous state which in turn is a result of its previous state. The MDP arouse to solve the infeasibility of storing all of this information.

2.3.4 Delayed Rewards

In reinforcement learning problems, the actions the agent performs on the environment determine not only its immediate reward, but also the next state of the environment(probabilistically). Thus, the agent must always take into account future states as well as the immediate reward when choosing among actions. The agent must be able to learn and then decide which of its actions are desirable based on rewards that can take place arbitrarily far in the future.

2.3.5 Q Learning

In Q learning, agent will be capable of knowing the expected reward of each action at every step. It will carry out a series of acts that will lead to maximum total reward. For the agent, this would effectively be a cheat sheet, making the agent aware of the correct actions to take. The total reward is called the Q-value and it can be formalized as[17]:

$$Q(s,a) = r(s,a) + \gamma maxQ(s,a) \tag{2.3}$$

The above equation states that the Q-value yielded from being at state s and performing action a is the immediate reward $r(s,a)$ plus the highest Q-value possible from the next states'. Gamma here is the discount factor which controls the contribution of rewards further in the future. Q learning is only practical for very small environments and quickly loses its feasibility when the number of states and actions in the environment increases. If an Imaginary environment with 10,000 states and 1,000 actions per state existed.A table of 10 million cells would be created. It is pretty clear that the Q-value of new states from already explored states cannot be inferred[17].

2.3.6 Deep Q Learning

To break the cycle and determine the ideal Q-value function, deep Q-Learning combines Q-Learning with a neural network. States are utilized as input and the optimal Q-value of all possible actions is used as output in the deep Q-Learning process. The following diagram illustrates the difference between Q-learning and Deep Q-learning techniques:

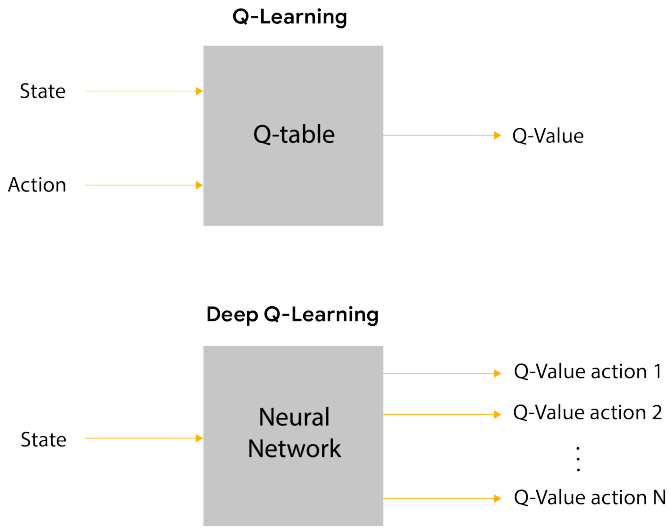


Figure 2.4: Difference between Q-learning and Deep Q-learning techniques

In Deep Q-Learning, the user stores all past experiences in memory and the future action defined by the output of Q-Network[17]. Thus, Q-network gains the Q-value at state S_t, and at the same time target network (Neural Network) calculates the Q-value for state S_{t+1} (next state) to make the training stabilized and blocks the abruptly increments in Q-value count by copying it as training data on each iterated Q-value of the Q-network. This process is presented in the image below.

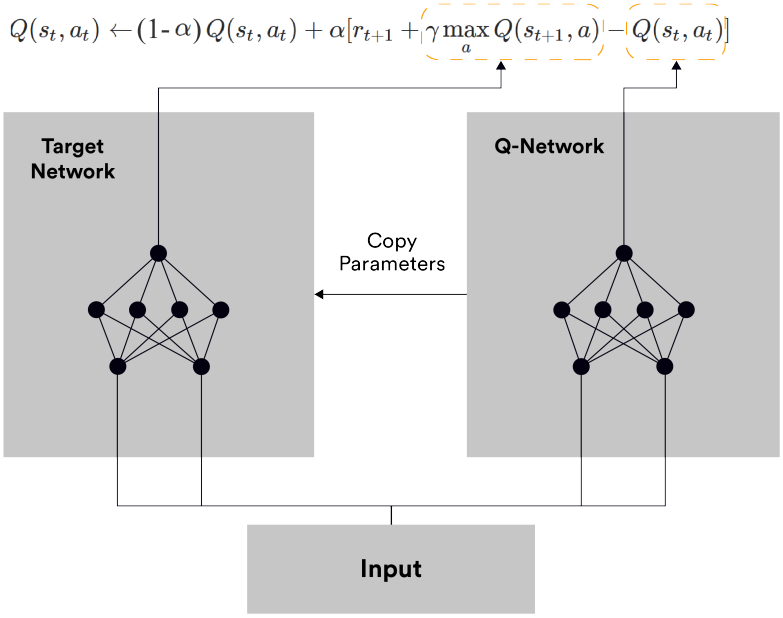


Figure 2.5: Deep Q Learning processes

2.3.7 Policy Gradients

When following a parametrized policy, the goal of a Reinforcement Learning agent is to maximize the expected reward. The challenge is to find parameters that maximize the reward. Gradient Descent is a common approach in Machine Learning Literature for tackling this maximizing problem[18].

Policy gradient methods are a subset of reinforcement learning techniques that depend on gradient descent to optimize parametrized policies in terms of expected return (long-term cumulative reward). Many of the issues that plague classic reinforcement learning systems, such as the lack of value function guarantees, the intractability problem deriving from unknown state information, and the complexity arising from continuous states and actions, are not present. Here comes the challenge, how do we find the gradient of the objective above which contains the expectation. Integrals are always bad in a computational setting. We need to find a way around them. First step is to reformulate the gradient starting with the expansion of expectation (with a slight abuse

of notation)[18].

The ambiguous state of information made the majority of the approaches presented within the reinforcement learning community inapplicable to the robotics and motor control applications. As a result, such systems must be treated as partially observable Markov decision problems, which often necessitates excessive computational demands. Most off-the-shelf reinforcement learning techniques cannot handle continuous states and actions in high-dimensional domains.

Policy gradient approaches differ greatly because they are not affected by these issues in the same manner. Continuous states and actions may be handled with in the same way as discrete ones, and learning performance is frequently improved as a result[18].

Assumptions and Notation

We assume that we can model the control system in a discrete-time manner and we will denote the current time step by k. In order to take possible stochasticity of the plant into account, we denote it using a probability distribution $x_{k+1} \sim p(x_{k+1}|x_k, u_k)|$ as model where $u_k \in \mathbb{R}^M|$ denotes the current action, and $x_k, x_k + 1 \in \mathbb{R}^N|$ denote the current and next state, respectively. We furthermore assume that actions are generated by a policy $u_k \sim \pi_\theta(u_k|x_k)|$ which is modeled as a probability distribution in order to incorporate exploratory actions; for some special problems, the optimal solution to a control problem is actually a stochastic controller[19].

The policy is assumed to be parameterized by $K|$ policy parameters $\theta \in \mathbb{R}^K$ .The sequence of states and actions forms a trajectory denoted by $\tau = [x_{0:H}, u_{0:H}]|$ where $H|$ denotes the horizon which can be infinite. In this article, we will use the words trajectory, history, trial, or roll-out interchangeably. At each instant of time, the learning system receives a reward denoted by $r_k = r(x_k, u_k) \in \mathbb{R}|$.

The general goal of policy optimization in reinforcement learning is to optimize the policy parameters $\theta \in \mathbb{R}^K$ so that the expected return

$$J(\theta) = E\{\Sigma_{k=0}^{H} a_k r_k\} \tag{2.4}$$

is optimized where a_k denote time-step dependent weighting factors, often set to $a_k = \gamma^k$ for discounted reinforcement learning (where γ is in [0,1]) or $a_k = 1/H$ for the average reward case.

For real-world applications, we require that any change to the policy parameterization has to be smooth as drastic changes can be hazardous for the actor as well as useful initializations of the policy based on domain knowledge would otherwise vanish after a single update step. For these reasons, policy gradient methods which follow the steepest descent on the expected return are the method of choice. These methods update the

policy parameterization according to the gradient update rule

$$\theta_{h+1} = \theta_h + \alpha_h \nabla_\theta J|_{\theta=\theta_h} \tag{2.5}$$

where $\alpha_h \in \mathbb{R}^+$ denotes a learning rate and $h \in \{0, 1, 2, \cdots\}$ the current update number. The time step k and update number h are two different variables. In actor-critic-based policy gradient methods, the frequency of updates of h can be nearly as high as of k . However, in most episodic methods, the policy update h will be significantly less frequent. Here, cut-off allows updates before the end of the episode (for $a_k = \gamma^k$, it is obvious that there comes a point where any future reward becomes irrelevant; a generically good cut-off point). If the gradient estimate is unbiased and learning rates fulfill $\Sigma_{h=0}^{\infty} \alpha_h > 0$ and $\Sigma_{h=0}^{\infty} \alpha_h^2 =$ const, the learning process is guaranteed to converge at least to a local minimum.

2.3.8 Proximal Policy Optimization (PPO)

The fundamental drawback of the policy gradient RL is the significant gradient variance. The advantage function is a common method for reducing variance in gradient estimates. The advantage function calculates how good an action is in comparison to the average for a given state.

PPO is a policy gradient approach in which the policy is explicitly adjusted. The goal is to lower the variance in the gradient estimate between the old and new policies. The RL algorithm's stability is enhanced by the minimization of variance. The advantage function can be written as follows:

$$A(s, a) = Q(s, a)V(s) \tag{2.6}$$

If the advantage function is positive, then it means action taken by the agent is good and we can a good reward by taking the action. So, the idea here is to improve those actions probability. On other hand, if the advantage resulted in negative, then we need to decrease the action probabilities[20].

One true issue with PPOs is that the advantage function can generate a noisy estimate, causing messy policy updates. The reason for this is that V(s) is another neural network, and due to the line search in gradient ascent, it can create a noisy estimate for the given state throughout the learning process.

2.3.9 Trust Region Policy Optimisation(TRPO)

TRPO introduces trust region techniques to RL instead of the line search approach. For the optimization process, the TRPO adds KL divergence constraints to enable the trust-region. It ensures that the new updates policy is not too dissimilar from the old policy, or that the new policy is within the old policy's trust region[20].

$$J^{CLIP}(\theta) = \mathbb{E}[min(r(\theta)A_{\theta_{old}}(s,a), clip(r(\theta), 1-\epsilon, 1+\epsilon)A_{\theta_{old}}(s,a))] \tag{2.7}$$

In the above equation, the function clip truncates the policy ratio between the range $[1-\varepsilon, 1+\varepsilon]$. The objective function of PPO takes the minimum value between the original value and the clipped value.

The optimal policy is determined in policy-based RL by changing policy directly, whereas the optimal policy is implicitly found in value-based RL by finding the optimal value function. In high-dimensional and stochastic continuous action spaces, policy-based RL is effective, as is learning stochastic policies. Value-based RL, on the other hand, outperforms in sample efficiency and stability.

2.3.10 Actor critic

Temporal differences learning is a process where the agent on each environment time step generates a learning estimate based on the immediate reward received and uses the information from the previous estimate and the difference between the temporally successive estimates. The main idea of temporal differences is, at every time step the values are updated to get closer to the same value in the next time step. Just like dynamic programming, updates are performed based on the current estimates without waiting for the final outcome.

Actor-Critic is a Temporal Difference (TD) version of Policy gradient. It has two networks: Actor and Critic. The actor decided which action should be taken and critic inform the actor how good was the action and how it should adjust. The learning of the actor is based on policy gradient approach. In comparison, critics evaluate the action produced by the actor by computing the value function.

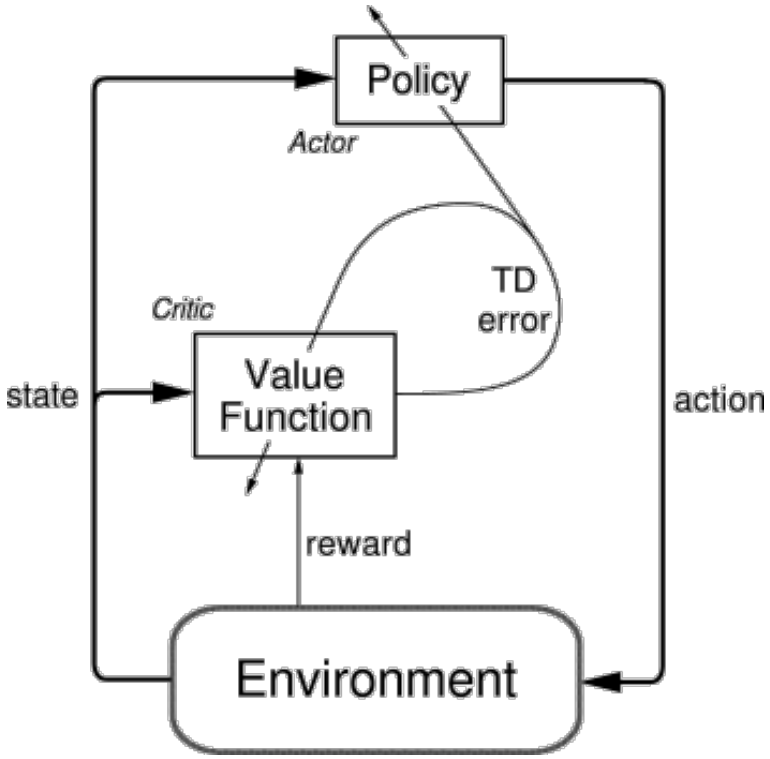


Figure 2.6: Actor-Critic architecture

This type of architecture is in Generative Adversarial Network (GAN) where both discriminator and generator participate in a game. The generator generates the fake images and discriminator evaluate how good is the fake image generated with its representation of the real image[21]. We can write the advantage function for actor-critic as follows:

$$A(s_t, a_t) = r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t) \tag{2.8}$$

2.4 Reinforcement Learning Framework The Problem

Understanding the strengths and weaknesses of each machine learning approach will save you a lot of time in Advance. Reinforcement Learning (RL) is used to solve sequential decisions problems. Generally in Reinforcement Learning, the agent's goal is to get maximum cummulative reward
not to get the immediate reward but reach the next state with the maximum value function - cumulative reward - to achieve this goal all Reinforcement Learning problems models are based on **Bellman recursive operation**[1].

2.4.1 Markov Decision Process

Markov decision problem theory and computation is based on using backward induction (dynamic programming) to recursively evaluate expected rewards, so the problem at hand must be described as Markov Decision Process "MDP".

2.4.2 Markov Decision process properties

1- Markovian Property

$$P(s_{t+1}|s_t) = P(s_{t+1}|s_{1,\dots,t}) \tag{2.9}$$

The future is independent of the past given the present. Which means the current state is enough to get the transition probabilities and immediate rewards following the action a.

2-Stationary

The physics - rules - of the environment don't change, the transition probability from states doesn't change, and the reward distribution function also doesn't change. The agent is the only element of change in the model.

2.4.3 Components of Markov Decision Process

Assuming finite MDPs, then Markov Decision Process is ideal to formally describe an environment for reinforcement learning and consist of the following components:

• State $s \in S$ where S is a finite set of states, and the state describes the environment.

• Actions $a \in A$, where A is a finite set of actions.

• Reward

$$R^a_{ss'} = r(s, a, a') \tag{2.10}$$

$$r_{t+1} = r(s_{t+1}, s_t, a_t) \tag{2.11}$$

• Transition model

$$P^a_{ss'} = P(s_{t+1}|s_t, a_t) \tag{2.12}$$

The probability of transition from state s to state s' when the agent takes action a.

• Discount Factor

$$\gamma \in [0, 1] \tag{2.13}$$

•Policy represents the "master plan" to solve the problem. It is represented by mapping from states into actions π(a|s) where for each state s it outputs the best course of actions

to follow.

•π^* is the optimal policy, the optimal mapping from states into actions and hence the optimal solution to an MDP.

2.5 Reward hypothesis

The term "reinforcement" comes from behavioral science. It refers to stimulus that is delivered immediately after a behavior to make it more likely to occur in the future. The goal of the agent can be expressed as maximizing the **total expected cumulative reward**.

If we are at time step t we will notice, the current time step reward and all rewards in the past have already been decided.Only the future rewards are in the agent's control. since the events in the future are the ones the agent can decide, we care about maximizing the **expected** return "return is the sum of rewards".

Utility "U": is the sum of rewards the agents observe in its **trajectory** -sequence of states transitions from the start state until the terminal state - but problem arises how can we determine which of two policies is better given infinite utility ?!!

Sequence of Rewards Assumptions

It's essential to specify how the agent takes the future into account when deciding action to take in the present, there are **three different models** to judge this subject.

• First, The finite-horizon model,

the agent has to optimise its expected reward for the next h steps, given the agent at time-step t = t_0.

$$E(\sum_{t=t_0}^{h} r_t) \tag{2.14}$$

When using the finite-horizon model the agents policy becomes function of both current state and remaining time-steps $\pi(a|s,h)$ The agent ends up taking different actions at the same state depending on how many time-steps are left and we lose the concept of **stationary** in the **Markovian Model** .

• Second, Average-reward model

$$\lim_{h\to\infty} E(\frac{1}{h}\sum_{t=t_0}^{h} r_t) \tag{2.15}$$

One problem of this model is no way to distinguish between two policies with the same utility -return - one policy gains a large amount of reward in first steps and the other gains large amounts of steps in later steps.

• Third, Infinite-horizon discounted model.

It takes all of the trajectory rewards into account with geometrically discount factor γ where $\gamma \in [0, 1]$

$$E(\sum_{t=t_0}^{\infty} \gamma^t r_t) \tag{2.16}$$

Discount factor γ can be seen as an interest rate, a probability of living another step, or mathematical trick to bound the infinite sum into finite and ensures the agent model will converge. In the case of infinite-discounted model the maximum return at each trajectory is

$$\frac{R_{max}}{1-\gamma} \tag{2.17}$$

2.6 Policies

A policy, representing the agent's behavior, mapping states to deterministic actions or probability distributions "the probability to take each action given state". The policy π can be

Stochastic Policy a $= \pi(a|s)$ Returns the probability of action executed in the given state.

Deterministic Policy a = $\pi(a|s)$ Tells the agent with certainty which action to take.

Generally the goal of the reinforcement learning is to find an optimal policy π^* helps the agent to choose the best course of actions to maximize total cumulative reward.

State Value function Value function [2] - also known as the state value function - yields the **expected** return if the agent starts in state S, then follows policy π, $V_\pi(s)$.

$$v_\pi = E_\pi[G_t|S_t = s] \tag{2.18}$$

$$V_\pi(s) = \sum_{a \in A} \pi(s|a)[R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_{\pi(s')}] \tag{2.19}$$

2.7 Bellman Equations

The value of any state in the MDP can be calculated as the sum of the immediate reward and the discounted value of the next state [2] .

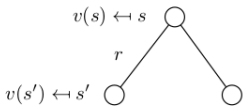
$$G_{discount} = R_{t+1} + \gamma.R_{t+2} + \gamma^2.R_{t+3} + \gamma^3.R_{t+4}..where \gamma \in [0, 1] \tag{2.20}$$

we know

$$v_\pi = E_\pi[G_t|S_t = s] \tag{2.21}$$

and using bellman recursive operation we have

$$V(s_t) = r(_{t+1}) + \gamma * v(s_{t+1}) \tag{2.22}$$



$$v(s) = \mathcal{R}_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'} v(s')$$

Figure 2.7: Bellman Equation

2.8 Action Value Function

Action value function Q(s,a) [2] , the agent start in state S takes action a and then follows policy π. **For** $s \in S$:

$$q_\pi(s, a) = E_\pi[G_t|S_t = s, A_t = a] \tag{2.23}$$

$$v_\pi(s) = q_\pi(s, \pi(s)) \tag{2.24}$$

2.9 Solving MDP

The goal of the reinforcement learning agent is to find an optimal policy ,determine the optimal policy π^* given the infinite-discounted horizon.

Interactions $\rightarrow$ rewards and value function $\rightarrow$ policy

$$v_\pi(s) = \sum_{a \in A} \pi(a|s)[R_s^a + \gamma \sum_{a \in A} P_{ss'}^a v_\pi(s^{'})] \tag{2.25}$$

The optimal value of state, is the reward the agent get start at state s and following the **optimal policy** π^* thereafter,

$$V^*(s) = max_\pi V^\pi(s) \tag{2.26}$$

Also optimal value function can be defined,

$$V^*(s) = max_a(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V^*(s')), s \in S. \tag{2.27}$$

$$\pi^*(s) = argmax_a(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V^*(s')), s \in S. \tag{2.28}$$

2.9.1 Dynamic Programming

Dynamic programming [1] denotes simplification in the reinforcement learning setting. We assume the agent has full knowledge of the environment(Markov Decision Process) that characteristics the environment. so it's easier than reinforcement learning when the agent knows nothing about the environment.

Policy Evaluation

Takes a policy π and returns an estimate to the state-value function V_π this approach the bellman update is applied to the state-value function until the value-function changes are nearly unnoticeable [1].

Policy Evaluation algorithm

Algorithm 1 Policy Evaluation [1]

Input: MDP, policy π small positive number θ
Output: V $\approx v_\pi$
Initialize V arbitrarily (e.g, V(s) = 0 for all s $\in S^+$)
repeat
$\triangle \leftarrow 0$
for s $\in S$ **do**
    v $\longleftarrow V(s)$
    V(s) $\leftarrow \sum_{a \in A(s)} \pi(a|s) \sum_{s',r \in R}$ p(s$'$, $r|s,a)(r + \gamma V(s'))$
    $\triangle \leftarrow$max($\triangle$,|v -V(s))**end for**
until $\triangle < \theta$

Policy Improvement

Takes an estimate V of the state-value function V_π corresponding to policy and returns an improved - or equivalent - policy π' , where $\pi' >= \pi for s \in S$. This algorithm first

constructs an action-value function estimate Q-table. Then for each state $s \in S$ the algorithm selects the action a which maximize Q(s,a),in other words $s \in S$ [1].

$$\pi^{'} = argmax_{a \in A(s)} Q(s,a) \tag{2.29}$$

Policy Improvement Algorithm

Algorithm 2 Policy Improvement [1]

Input: MDP, value function V
Output: policy $\pi^{'}$
for $s \in S$ **do**
    for $a \in A(s)$ **do**
        Q(s,a) $\leftarrow \sum_{s^{'} \in S, r \in R} P(s^{'}, r|s,a)(r + (s^{'}))$
    end for$\pi^{'} \leftarrow argmax_{a \in A(s)} Q(s,a)$
end for
return π

Policy **Iteration**

Policy iterations proceeds as a sequence of policy evaluation and improvement steps, and is guaranteed to converge to the optimal policy after infinite interactions with the environment (for an arbitrary finite MDP) [1].

Policy Iteration Algorithm

Algorithm 3 Policy Iteration [1]

Input: MDP, small positive number θ
Output: policy $\pi \approx \pi^{*}$
Initialize π arbitrarly (e.g, $\pi(a|s) = \frac{1}{A(s)}$ for all $\in S$ and $a \in A(s)$)
policy-stable $\leftarrow$ false
repeat
    $V \leftarrow$ **Policy Evaluation**(MDP, π, θ)
    $\pi^{'} \leftarrow$ **Policy Improvement**(MDP, V)
    if $\pi = \pi^{'}$ **then**
        policy-stable $\leftarrow$ true
    end if
    $\pi \leftarrow \pi^{'}$
until `policy-stable = true`
return π

Value **iteration**

Takes as input policy π to estimate the state-value-function V_{π} and returns an improved policy, in this approach every sweep over the state space simultaneously performs policy evaluation and policy improvement [1].

Value iteration Algorithm

Algorithm 4 Value Iteration [1]

Input: MDP, small Positive number θ
Output: policy $\pi \approx \pi^*$
Initialize V arbitrarily (e.g V(s) = 0 for all $s \in S^+$)
repeat
    $\triangle \leftarrow 0$
    for $s \in S$ **do**
        $v \leftarrow V(s)$
        $V(s) \leftarrow max_{a \in A(s)} \sum_{s' \in S, r \in R} P(s', r|s, a)(r + \gamma V(s'))$
        $\triangle \leftarrow max(\triangle, |v - V(s)|)$
    end for
until $\triangle < \theta$
$\pi \leftarrow$ **Policy Improvement(MDP,V)**
return π

2.9.2 Model-Free Reinforcement

Tabular Methods In the previous section we discussed the methods of obtaining the optimal policies for an MDPs with assumption the model its well-known - states transition probabilities and the reward function R(s,a)- Most of the time the model is unknown and the agent must interact with the environment directly to obtain the information, model free allows learning a policy - controller - without learning the model.

Interactions $\rightarrow$ rewards $\&$ value function $\rightarrow$ policy

Monte-Carlo Methods

Monte Carlo [22](**MC**) works by averaging the returns the agent receives from the environment on an episodic basis.

Note Reinforcement learning tasks can either be episodic with well defined ending points - like end of the game, crash of car - or continuous ones with no ending points like modeling the economy of some country. In episodic task every state-action pair is called **visit**.

Monte-Carlo Prediction First Visit for Action Values:

Temporal Difference Learning

Unlike MC methods that wait until the end of the episode to learn, TD [23] learning the agent is not allowed to take a break for evaluation and improvement, TD Learning waits for only one time-step and uses temporal errors to update the value.

TD(0) prediction

Algorithm 5 Monte-Carlo Prediction First Visit for Action Values [1]

Input: policy π, positive integer num-episodes
Output: value function Q($\approx q_\pi$ if num-episodes is large enough)
Initialize returns-sum(s,a) = 0 for all $s \in S$, $a \in A(s)$
Initialize returns-sum(s,a) = 0 for all $s \in S$, $a \in A(s)$
for $i \leftarrow 1$ `to num-episodes` **do**
    Generate an episode $S_0, A_0, R_1, ..., ST$ using π
    for $t \leftarrow 0$ `to` $T-1$ **do**
        if (S_t, A_t) `is first visit (with return` G_t`)` **then**
            $N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$
            return-sum$(S_t, A_t) \leftarrow$ return-sums$(S_t, A_t) + G_t$
        end if
    end for
end for
Q(s,a) $\leftarrow$ return-sum(s,a)/N(s,a) for all $s \in S, a \in A$
return Q

Algorithm 6 TD(0) [1]

Input: policy π, positive integer num-episodes
Output: value function V($\approx v_\pi$ if num-episodes is large enough)
Initialize V arbitrarily (e.g V(s) = 0 for all $s \in S^+$)
for $i \leftarrow 1$ `to num-episodes` **do**
    Observe S_0
    $t \leftarrow 0$
    repeat
        Choose action A_t using policy π
        Take action A_t and observe R_{t+1}, S_{t+1}
        V$(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$
        $t \leftarrow t + 1$
    until S_t is terminal
end for

2.10 Deep Learning

Deep learning has turned out to be broadly applicable technique. It has been found useful in everything from hedge-fund algorithms to bioinformatics. Deep Learning techniques utilize the same building blocks as simpler neural networks.

2.10.1 Biological Basis

as Amr Awadallah - CTO of Cloudera - says" the most important skill in the 21th century is to learn how to learn", the human brain is the most incredible computational device in

existence, its true that fast – quantum – microprocessors are faster, but human brain has the ability to adapt to new situations, learn new skills do innovation and design thinking unsurpassed by any known machine. Each nerve cell in the brain is know as neuron. Neurons in the brain form a networks, by connections known as synapses.

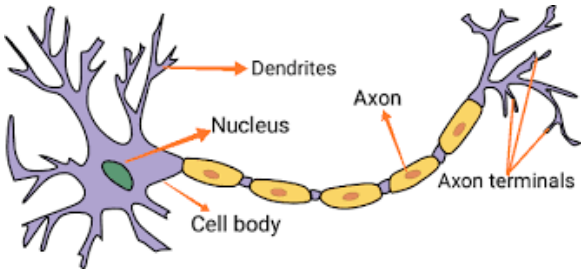


Figure 2.8: Biological Neural Network

2.10.2 Artificial Neural Network

Feed-Forward Neural Networks The signal generally moving in one direction through the networks, followed by back-propagation which determines errors at the end of each input signal through the structure and distribute the error back through the network's weights.

Neurons the smallest unit in neural networks, it holds a vector of weights. At each time-step a vector of input Is passed to the neuron. The neurons apply dot product operation to the input and its weights then gives the result of the dot product into an activation function, usually the activation function is non-linear transformer – to allow the neural network to generalize to non-linear situations

Layers neurons in feed-forward neural networks are organized in layers, each layer consist of a certain number of neurons, in feed-forward neural networks the signal will pass from one layer into the following. Every neuron in each layer is connected to every neuron in the previous layer. The first layer is known as the input layer. It receives signals from some external source, the last layer is known as the output layer, layers in between are known as hidden layers.

Back-propagation

back-propagation finds the error in neural networks outputs and uses the error to modify the weights of the neural networks so the neural network can generalize well in the future, we calculate the error between the neural networks output for some input and expected output, the partial derivative of the output neurons activation function is applied to all the weights of the previous layers, we calculate the partial derivative of error with respect to all the hidden layers weights to determine how much each neuron was responsible for the incorrect output in the output layer Uses Multi-chain rule and gradient descent calculates the partial derivative of the error with respect to all the

hidden layers weights to determine how much each neuron was responsible for the incorrect output in the output layer so optimizing the weights of the neural network.

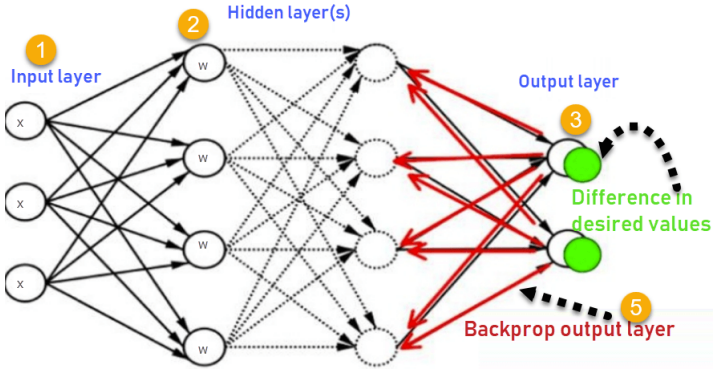


Figure 2.9: Backpropagtion neural network

2.11 Deep Reinforcement Learning

We have covered model-free reinforcement learning and Deep learning, now its time to combine both and get deep reinforcement learning. The tasks that reinforcement learning solves are mostly in domains in which the agent observes low-dimensional state spaces, however using function approximation - as deep learning- allows the agent to be successful at learning policies directly from high-dimensional state input.

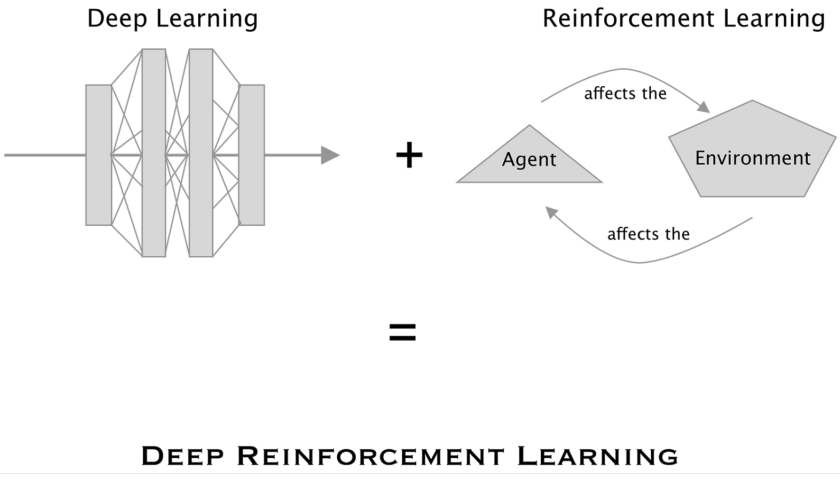


Figure 2.10: Deep Reinforcement Learning

2.11.1 Deep Q-networks

Deep Q-networks [24] algorithms outperform human experts in Atari games, as we have already discussed Q-learning refers to solving a control task by learning a function called Q-function, Q-learning is an off policy method - means with infinite amount of samples from the environment,the choice of the policy will not affect reach

the true action-value function it will only allows us to reach the optimal action-value function faster- Q-function must uses data to learn how to make accurate predictions about the action value function

$$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \gamma * (R_{t+1} + \gamma * max_a Q(s_{t+1}, a) Q(s_t, a_t)) \tag{2.30}$$

There are **three main techniques** Deepmind used in Deep Q-network.
let's assume the example of GridWorld, in which we have an agent and it tries to reach the goal as fast as possible, and at all cost avoid ending up at the pit, and there is a wall somewhere in the world, at each time step the agent is allowed to take one of the following actions [right, left, up, down] [25].

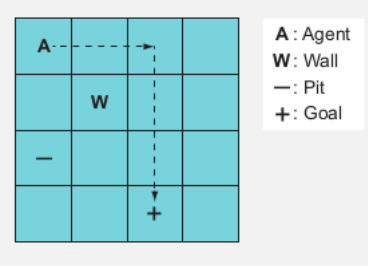


Figure 2.11: Grid World Environment

First Technique instead of passing the state and the action to the q-network we just pass the state and get output from the neural network with length equal to the number of possible actions so the function now can be assigned as $Q_A(s)$. each of the vector output represent the action value function for the corresponding discrete action – the total expected reward start at state s taking action a observe reward r and following policy π there after- in the below example there are four possible actions *up, left, right, down* [25].

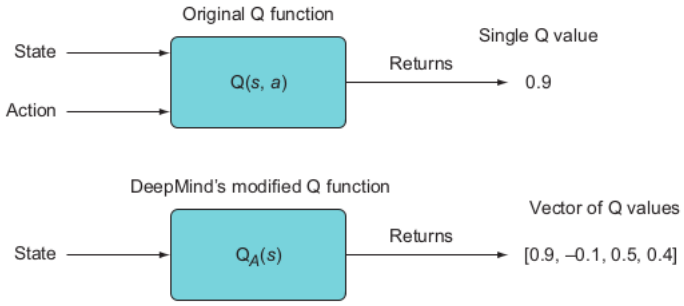


Figure 2.12: Deep Mind Q_function

Second Technique Experience Replay unless at all episodes the environments were initialized to the same states and locations the agent fail to solve the problem, this problem is known as catastrophic forgetting – the agent just end up memorizing the

states and if the game started at random locations each episode then the agent will fail. Actually catastrophic forgetting is a very important issue with gradient descent methods in online-learning. **online-learning** is back-propagating the loss after each move as we play the game. ***Note*** We generally don't face catastrophic forgetting with supervised learning since we do randomized batch learning where we don't update the weights until we have passed through some random set of training data and computed the sum of average gradient for the batch.

Experience replay provides us with batch updating in an online-learning, and allows us to randomize over data to remove correlation between observations sequences and smooth over the changes in the data distribution. ***Steps***

*in state s we take action a observe new state $s + t + 1$ and reward r_{t+1}

* store the tuple $(s_t, a_t, r_{t+1}, s_{t+1})$ * continue to store experiences in this list untill we have filled the replay buffer

* once the replay buffer is filled randomly select subset (mini-batch gradient descent)

* pass through this subset and calculate the targets for each subset element

$$R_{t+1} + \gamma * max_a Q(s_{t+1}, a) \tag{2.31}$$

use the targets and the old predicted values $Q(s_t, a_t)$ for batch training

$$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \gamma.(R_{t+1} + \gamma * max_a Q(s_{t+1}, a) Q(s_t, a_t)) \tag{2.32}$$

Third Technique Target Q-Network

the action-value function update equation is

$$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \gamma.(R_{t+1} + \gamma * max_a Q(s_{t+1}, a) Q(s_t, a_t)) \tag{2.33}$$

If we keep updating the parameters of the $Q - network$ after each subset of replay buffer, this might cause instability [2]. this is even worse than chasing a moving target. In $Q - learning$, we are bootstrapping - making predictions out of predictions - the iterative updating of the target action-value network after a period will reduce the correlations with the target [2].

2.11.2 Policy Gradients methods "Learning to pick the best policy"

Training Q-network outputs action values and then we can use epsilon greedy policy on the top of the action values, what if we skip selecting policies on top of DQN using policy gradient we instead train neural network to output policy directly, this neural

Algorithm 7 Deep Q-Learning Algorithm [2]

Initialize replay memory D with capacity N
Initialize $action - value$ function $\hat{q}$ with random weights w
Initialize $target - value$ weights $w^{-} = w$
for `the episode e = 1 to M` **do**
  Initial input frame x_1
  Prepare initial state: S = $\phi(< x_1 >)$
  for `time step t = 1 to T` **do**
    Sample
    choose action A from State S using policy $\pi = \epsilon - greedy(\hat{q}(S, A, w))$
    Take action A, observe reward R and next input frame x_{t+1}
    Prepare next state: $S^{'} = \phi(< x_{t-2}, x_{t-1}, x_t, x_{t+1} >)$
    Store experience tuple$(S, A, R, S^{'})$ in the replay memory D
    $S = S^{'}$
    Learn
    Obtain random mini-batch of tuples (S_j, a_j, r_j, s_{j+1}) from D
    set target $y_j = r_j + \gamma max_a \hat{q}(s_{j+1}, a, w^{'})$
    update: $\triangle w = \alpha(y_j - \hat{q}(s_j, a_j, w)).\nabla_w q(s_j, a_j, w)$
    Every C steps reset $w^{-} = w$
  end for
end for

network called policy network, policy network tells us exactly what to do given the state we're in. all we need to do is randomly sample from the probability distribution $P(A|S)$. Policy-based methods are better for continuous action spaces than value-based methods, because in discrete space we can easily generate an action vector then pick the action with the maximum value but for continuous case the ***max*** operation turned out into an optimization problem.

$$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \gamma * (R_{t+1} + \gamma * max_a Q(s_{t+1}, a) Q(s_t, a_t)) \tag{2.34}$$

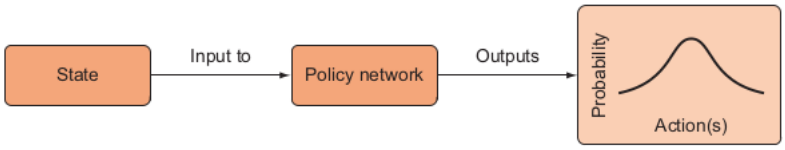


Figure 2.13: Policy Gradient Sampling

The most beneficial actions will have the highest chance of being selected from random sampling, since they are assigned the highest probability.

Policy function approximation

Function approximation can extend policy-based methods to cover large and continuous state spaces. To do so, we optimize the policy with parameters θ.

Discrete spaces

$$a = \pi(s, a, \theta) = [a|s, \theta] \tag{2.35}$$

for example, an approximation function would simply be the linear combination of features.

$$f(s, a, \theta) = x(s, a)^T.\theta \tag{2.36}$$

by applying a soft-max function, we can then turn this into a probability distribution. Only works for a discrete set of actions.

Continuous spaces

$$f(s, a, \theta) = x(s, a)^T.\theta \tag{2.37}$$

$$a = \pi(s, a, \theta) = N(\mu, \sigma^2) \tag{2.38}$$

we can determine the parameters of our Gaussian by

$$\mu = f(s, a, \theta) \tag{2.39}$$

and fix the variance σ^2

Stochastic policy gradient

The policy outputs vector representing probability distribution over actions,in the first episodes the agent can select two different actions from the same state, but since the env policy function approximation function approximation – neural network - produces an action vector representing a probability distribution over all possible actions. The policy we follow is selecting from this probability distribution which means if the agent ends up in the same state twice it may select two different actions and this particularity is good in case of aliased states. But since the environment is stationary the policy will eventually converge.

Define an objective Function and Reinforcing good actions

The policy network denoted by π parameterized by parameters θ ,which represents weights of the neural network, whenever we run a forward pass, the parameter θ is fixed, the variable is the the data – state – we pass to the network. Hence we donate the policy as π. Given the state we are in. We encourage the policy to take actions that maximize the total cumulative reward[25]. We want to make smooth updates to our gradients to encourage the network to assign more portability to these winning actions in the future. We have to make the updates small because we want stochastic action selection to ensure exploration, also we want to be able to weight how much we assign credit to each action in the trajectory. since the last actions contributed more to the reward more than the first action in the episode.

Our objective function

$$\gamma_t * G_t * \log \pi(a|\theta) \tag{2.40}$$

this $\pi(a|s,\theta)$ notation makes it clear the probability of selecting an action depends on the network parameters, and because all the probabilities must sum to one then maximizing probability of good action a= $\pi(a|s,\theta)$ - will decrease the probabilities of other actions. So the agent seeks to maximize $\pi(a|s,\theta)$, one problem with probabilities objective function is they are bounded between 0 and 1 so the range of value the optimizer can operate is very limited, so we use instead The surrogate objective function $\log \pi(a|s,\theta)$ has large dynamic range (-∞, θ) this allows the optimizer to operate in larger range, still we have to solve the problem of credit assignment, so we add parameter G_t and γ_t , we want to discount more distant actions more than more recent ones

$$G_t = r_t + r_{t+1} ... + rT - 1 + r_T \tag{2.41}$$

$$\gamma_t = \gamma(Tt) \tag{2.42}$$

2.11.3 Actor Critic (Temporal Difference update)

Since MC -Monte Carlo Methods - are not sample efficient people generally use bootstrapping - estimate the value using estimates- instead of updating $V(S)$ by Monte-Carlo Sampling, and get advantage function like

$$\delta \longleftarrow R - V(s) \tag{2.43}$$

The Advantage Function when bootstrapping with temporal difference update

$$\delta \longleftarrow r_t + \gamma.V_w(s_{t+1}) - V_w(s_n) \tag{2.44}$$

Baseline with temporal Difference is called ***Actor Critic***

Algorithm 8 Actor Critic (s_0, π_0)

Initialize π_θ to anything
Initialize Q_w to anything
loop `For each Episode`
    Initialize s_0 and set $n \longleftarrow 0$
    while `s is not terminal (for each time step t)` **do**
        Sample a_t $\pi_\theta(a_t|s_t)$
        Execute a_t an observe s_{t+1}, r_t
        A(s,a)$\longleftarrow r_t + \gamma.V_w(s_{t+1}) - V_w(s_t)$
        Update value network: $w \longleftarrow w + \alpha_w.\gamma^t.A(s,a)\nabla_w V(s_t)$
        Update Policy network: $\theta \longleftarrow \theta + \alpha_\theta.\gamma^t.A(s,a).\nabla_\theta \log \Pi_\theta(a_t|s_t)$
    end while
end loop
Return π_θ

2.12 Related Work

The work in [26] provided a novel technique for making the Radio Access Network (RAN) more energy efficient, based on a realistic traffic profile and needed throughput. There is also a need for a metric that quantifies the link between Energy Efficiency (EE), Spectrum Efficiency (SE), and intended throughput.

The proposed method for reducing RAN energy consumption defines the available throughput in order to determine how much energy is necessary. The RAN's energy consumption may be lowered by simply switching (OFF/ON) Base Stations (BSs) based on the needed throughput, resulting in an energy efficient RAN. However, it did not provide a standardized environment, nor did it provide a more intelligent or scalable solution to the problem.

Chapter Three

System and Environment Modelling

3.1 Available Environments

It is impossible to develop the ideal operation strategy for controlling RAN base stations by physically conducting tests, owing to the enormous costs of such a system in the first place. The high cost of failure is another factor that prevents physical experimentation. For those two reasons, research in this subject generally begins by modeling various scenarios in software designed particularly for simulation, and then, after demonstrating satisfactory results, practical trials are conducted.

There a lot of simulators which can easily replicate what can occur at a RAN (Radio Access Network), but all of them come up short when integrating them with python and deep learning libraries, a disadvantage that greatly hinders the objective of this thesis. Some of the simulators tried at this research:

- **OMNeT++**: (Objective Modular Network Testbed in C++)[27] is a C++ simulation toolkit and framework that is mostly used to create network simulators. For non-commercial simulations, such as at academic institutions and for education, OMNeT++ is available for free. OMNEST is a commercially available version of OMNeT++.

  OMNeT++ is a simulation framework that does not include models for network protocols such as IP or HTTP. Several external frameworks include the core computer network simulation models. The most widely used is INET, which provides models for a wide range of network protocols and technologies, including IPv6, BGP, and others. In addition, INET provides a collection of mobility models that may be used to mimic node movement in simulations. The LGPL or GPL licenses apply to the INET models. The topology description language of OMNeT++ is NED (NEtwork Description).

  Additional tools, such as Python-based tools, have been created to manage and decrease the time it takes to run large-scale simulations. However, it lacks the requisite

integration for deep learning.

- **NS-3**: Network Simulator 3[28] is written in C++ and Python and supports scripting. Python wraps the ns library with the pybindgen package, which delegated the parsing of the ns C++ headers to castxml and pygccxml to build the necessary C++ binding glue. These automatically generated C++ files are then built into the ns Python module, allowing users to utilize Python scripts to interface with the C++ ns models and core. The ns simulator has an integrated attribute-based system for managing simulation parameter default and per-instance settings. It has the same drawback as OMNET++ which it does not have an integration for deep learning libraries.

- **GloMoSim**: Global Mobile Information System Simulator[29] is a network protocol simulator that can mimic both wireless and wired network systems.

  GloMoSim is based on Parsec, a C-based parallel programming language that allows for parallel discrete event simulation. At the moment, GloMoSim only supports protocols for a wireless network. The simulation protocols are compiled using the Parsec compiler. GloMoSim is no longer under development and is currently deprecated.

- **LTESIM**: G. Piro and F. Capozzi of the ”Politecnico di Bari” created LTE-Sim[30], an open source framework for simulating LTE networks. It includes the Evolved Universal Terrestrial Radio Access (E-UTRAN) and the Evolved Packet System, as well as other features of LTE networks (EPS). Single and heterogeneous multi-cell settings, QoS management, multi-user environments, user mobility, handover processes, and frequency reuse techniques are all supported. User equipment (UE), evolved Node B (eNB), Home eNB (HeNB), and Mobility Management Entity/-Gateway (MME/GW) are the four types of network nodes that are represented. At the application layer, four distinct traffic generators have been created, and data radio bearer management is provided. Finally, AMC system, Channel Quality Indicator feedback, frequency reuse approaches, and physical layer models have been established.

- **OPNET**: OPNET Network Simulator[31] is a program that may be used to model the behavior and performance of any network. Opnet Network Simulator stands out from other simulators because of its strength and adaptability. IT Guru offers pre-built protocols and device models. It enables you to design and simulate various network topologies. You cannot build new protocols or change the behavior of current ones since the collection of protocols/devices is fixed.

Those simulators have common problems that prevent us from conducting the experiment of the thesis which are the following:

- The lack of integration with python, as that means that no model can be implemented and used to control the simulation settings in any of those simulators.
- Scheduled blackouts can not be implemented or executed in either of the simulators, rendering simulation of unreliable situations impossible.
- Even if power usage simulators (e.g. HOMER[32]) are used it proves impossible to integrate between RAN simulator and power consumption simulator and achieve near real-time simulation

3.2 System Model

In this section the system model behind the proposed methodology is presented and discussed. The proposed methodology is based on characterizing the throughput within the RAN. Using a small number of Base Stations means that the Signal to Interference Ratio is will be high. As the number of Base Stations starts to increase in the network this will result in higher interference received by the Mobile Station lowering the Signal to Interference Ratio (SIR). This indicates that the RAN operation's offered throughput will be reduced, and thus boosting the RAN operation would cost more energy. The throughput per user is calculated using the SIR experienced at the Mobile Station. Following that, the RAN offered throughput is estimated to determine how many Base Stations are necessary to cover the RAN desired throughput adequately. After that, further performance measures are utilized to quantify energy consumption and to evaluate the energy efficiency of different RAN processes.
In this case a RAN model of 16 Base Stations is used as the model of this research.

3.2.1 Propagation Model

The power that a Mobile Station can receive from a Base Station is governed by the equation:

$$P_r = P_t - L_p \quad \text{(db)} \quad (3.1)$$

Where P_r is the power received at the MS (Mobile Station), P_t is the power transmitted from the BS (Base Station), and L_p represents the path loss also called propagation loss[26].
In this thesis, Equation 3.1 is used with the assumption that all internal losses in the receiver (Mobile Station) and transmitter (Base Station) terminals are included in P_r and P_t.

Equation 3.2 is used as the propagation loss model in this thesis to treat the received signal in relation to a propagation loss model that is consistent with the proposed RAN scenario. This propagation loss model was chosen due to its relevancy with the proposed system assumption as presented in the next sections.

$$L_p = PL_b + PL_{tw} + PL_{in} \quad \text{(db)} \quad (3.2)$$

Where PL_b represents the basic propagation loss, PL_{tw} the loss through wall, and PL_{in} and the indoor loss[26].
Where each of the basic propagation loss, the loss through wall, and the indoor loss are calculated with the following equations respectively in the set of equations 3.3.

$$\begin{aligned} PL_b &= 35log_{10}(d) + 15.2 \\ PL_{tw} &= 20 \\ PL_{in} &= 0.5 \end{aligned} \quad (3.3)$$

Where d represents the distance between the BS (Base Station) and the MS (Mobile Station) measured in meters[33] [26].
By applying each of the equations 3.3 into its respective place we get the final path loss equation used in the thesis:

$$L_p = 35log_{10}(d) + 35.7 \quad \text{(db)} \quad (3.4)$$

As stated below in equation 3.5, the distribution of propagation loss values for a large range of distances tends to be a log-normal distribution, where the propagation model is a function of the distance d between the transmitter and receiver (downlink)[26].

$$L_p(d) = \overline{L_p}(d) + \chi_\sigma \quad \text{(db)} \quad (3.5)$$

Where X_σ is zero mean log-normally distributed random variable with standard deviation σ in dB, $\overline{L_p}(d)$ the arithmetic mean value of the propagation loss as a function of distance.
The standard deviation for the propagation loss model in equation 3.4 is 7 dB, with the mean value of the computed propagation loss being log-normally distributed with a zero mean and a standard deviation of 7 dB to account for the shadow fading induced in the RAN operation.

In this thesis the main concern is the channel capacity, to calculate the channel capacity first, the normalized channel capacity has to be calculated as shown in the equation 3.6[26].

$$\tilde{C} = log_2(1 + \frac{S}{\gamma + I}) \qquad \text{(bit.sec}^{-1}\text{.hz}^{-1}\text{)} \quad (3.6)$$

Then the channel capacity can be calculated given the normalized channel capacity and the bandwidth which the Base Stations in the RAN use as described in the equation.

$$C = W.\tilde{C} \qquad \text{(bit.sec}^{-1}\text{)} \quad (3.7)$$

plugging equation 3.6 into equation 3.7 we get the Shannon-Hartley theorem that states that that if S is the average transmitted power, and the noise is white thermal noise of the power γ in the channel bandwidth of W, Then it is possible to receive binary digits with the highest bound of a rate of C as shown in equation 3.8.[26]

$$C = W.log_2(1 + \frac{S}{\gamma + I}) \qquad \text{(bit.sec}^{-1}\text{)} \quad (3.8)$$

Equation 3.6 is employed in this thesis under the bandwidth normalization condition, with one exception: cellular systems are interference-limited. As a result, the interference dominates the noise, and the channel capacity model becomes equation 3.9.[26]

$$\tilde{C} = log_2(1 + \frac{S}{I}) \qquad \text{(bit.sec}^{-1}\text{.hz}^{-1}\text{)} \quad (3.9)$$

Which results the equation that describes the channel capacity of the cellular network that interference-limited, as shown in the equation 3.10

$$C = W.log_2(1 + \frac{S}{I}) \qquad \text{(bit.sec}^{-1}\text{)} \quad (3.10)$$

As an example, applying those equation to a RAN of one base station yields results like what figure 3.1 shows.[26]

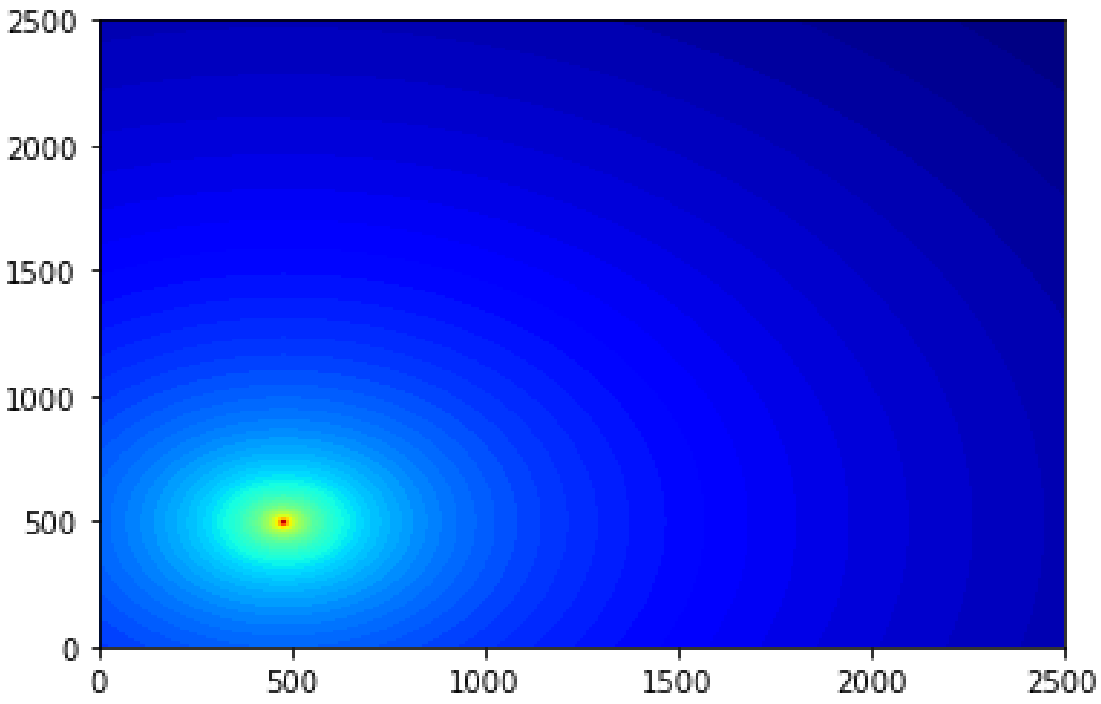


Figure 3.1: Example of Path Loss Model

3.2.2 Assumptions

Some assumptions are made before specifying the parameters and circumstances assumed for the suggested methodology's procedure. The assumptions and system parameters are provided and explored in this part, followed by the suggested methodology's algorithm.

The RAN's operation area is thought to be in a densely populated region. Because a RAN serving a dense metropolitan region inherently delivers higher throughput than a RAN serving a rural area, it is predicted to consume more energy, making it a common testing situation for most technologies aimed at reducing RAN energy usage. This is because a higher number of active users necessitates a higher RAN throughput, resulting in higher energy consumption.

Furthermore, because the area is densely populated, the RAN provided throughput is the most important consideration in network dimensioning. As a result, it is the most important element impacting network performance.

The RAN provided throughput, as indicated in, is the most important aspect in determining the GoS in micro/Pico cells. In other words, in dense metropolitan locations where the demand for provided throughput outnumbers the need for RAN coverage, supplied throughput becomes the most important element influencing the RAN GoS. This assertion supports the thesis's overall approach, which focuses less on the RAN coverage issue.

Another assumption assumed in this thesis that all base stations in RAN (Radio Access Network) are connected to a global electricity grid and this grid is unstable. To stabilize the grid some of the base stations are cut-off from it in a pre-determined schedule

3.3 Environment Implementation

The OpenAI gym[34] is a place where learning agents may be developed and tested. It is centered on reinforcement learning agents and is best suited for them, but it does not exclude the employment of other methods such as hard programmed game solvers or other deep learning approaches.

Reinforcement learning (RL) is a machine learning discipline that deals with decision-making and motor control. It investigates how an agent may learn to attain objectives in a complicated and unpredictable environment. It's thrilling for two reasons:

- RL is very general, covers any problem that requires a series of decisions, such as operating a robot's motors so that it may run and leap, making business decisions such as pricing and inventory management, or playing video games and board games. Even supervised learning issues with sequential or organized outputs can benefit from RL.
- In a variety of challenging contexts, RL algorithms have begun to show promise. RL has a long history, but it needed a lot of problem-specific engineering until recent improvements in deep learning. DeepMind's Atari results, Pieter Abbeel's BRETT, and AlphaGo all employed deep RL algorithms that didn't make too many assumptions about their environment and could thus be utilised in various situations.

RL research, on the other hand, is hampered by two factors:

- Better benchmarks are required. Large labeled datasets like ImageNet have fueled advancements in supervised learning. A wide and diversified assortment of habitats would be the closest analogy in real life. However, the present open-source RL environment collections lack diversity, and they are frequently difficult to set up and operate.
- There is a lack of consistency in the contexts utilized in publications. Small changes in the issue specification, such as the reward function or the list of activities, can substantially change the complexity of a job. This problem makes reproducing published research and comparing results from multiple studies difficult.

These two problems are effectively solved by the standardized environment implementation that open-ai gym offers. Open-ai gym allows to use from various

pre-implemented environments and is also flexible enough to allow implementing a custom environment.

Open Gym Environment

class Open Gym Environment

variables:

$action\_space$ (OpenGym type): The type and dimension of the action space.

$observation\_space$ (OpenGym type): The type and dimension of the observation space (state space).

functions:

$step$: This function takes an action and performs in on the current state of the environment which transitions the environment to the next state.

Input:

- action: the action proposed based on the current state (observation) of the environment, type of the variable depends on $action\_space$.

Output:

- observation: the next state (next observation) resulting from applying the action on the current state, type of the variable depends on $observation\_space$.
- reward (Number): a measure of the action taken which tell whether the action done on the current state (observation) is good or bad.
- done (Boolean): indicates if an episode is done or not.
- info (dictionary): info that can be useful for debugging purposes.

$reset$: resets the environment and return it to the initial state.

Input:

- none.

Output:

- observation: the initial observation of the environment, type of the variable depends on $observation\_space$.

$render$: renders the current state of the environment.

Input:

- mode: sets the rendering mode of the environment.

Output:

- none.

$close$: terminate the environment instance and deletes it.

Input:

- none.

Output:

- none.

end class

As shown in the block. There an backbone API for each open-ai gym environment which has the basic features that every environment must have to be fully operational, those features in summary are:

- The type and form of the $action\_space$ must be described in the constructor, which will include all of the actions that an agent may do in the environment. Similarly, the $observation\_space$ has to be defined, which contains all of the data in the environment that the agent will observe.
- The $reset$ function will be used to return the environment to its default state on a regular basis. The model will then supply an action that must be done, and the next observation will be returned after several $step$ function calls across the environment. This is also where rewards are computed.
- Finally, the $render$ method may be used to print a depiction of the environment on a regular basis. It might be something as basic as a print statement or something as complex as displaying a 3D projection.

The typical ”agent-environment loop” is simply implemented in the way shown in figure 3.2. The agent picks an action each time-step, and the environment responds with an observation and a reward.

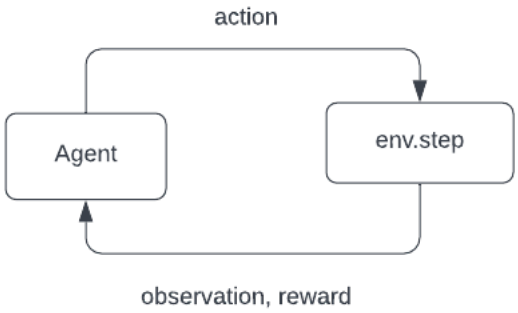


Figure 3.2: Environment Cycle

In order to implement the environment needed to simulate any RAN network with the required integration to copulate the environment with any deep reinforcement learning agent or any kind of agents, open-ai gym environment is a good start to implement on top of it. Also other helper functions that deal with path loss, calculating signal and interference, SIR (Signal to Interference Ratio), and bit rate are to be implemented first.

Environment Class

class RAN Environment

variables:

bs_loc (Array of Coordinates): Positions of each base station in the RAN.

bs_power (Array of Numbers): Initial power values of the base stations measured in dbm.

$electricity$ (Array of Booleans): Electricity status of the base stations, 0 represents Available and 1 represents blackout.

$bs\_blackout\_start$ (Array of Numbers): Array of starting times of blackouts.

$bs\_blackout\_end$ (Array of Numbers): Array of ending times of blackouts.

max_power (Number): Maximum power that a base station can emit measured in watts.

min_power (Number): Minimum power that a base station can emit measured in watts.

$n\_actions$ (Number): number of actions available per base station.

$grid\_width$ (Number): Width of grid that the RAN operates on measured in meters.

$grid\_height$ (Number): Height of grid that the RAN operates on measured in meters.

$bitrate$ (Matrix of Numbers): The bitrate of each point on the grid up to specific resolution ($1m^2$).

$reward$ (Number): The reward achieved from the latest action.

$actions$ (Array of Numbers): All actions can be taken per base station. Represents the power values that a base station can take.

n_bs (Number): Number of base stations in the RAN network.

$episode$ (Number): Current time/episode of the environment.

$limit$ (Number): Number of episodes per step.

$step\_size$ (Number): Number of episode to jump after taking an action.

$action\_space$ (OpenGym type): The type and dimension of the action space.

$observation\_space$ (OpenGym type): The type and dimension of the observation space (state space).

functions:

$step$: This function takes an action and performs in on the current state of the environment which transitions the environment to the next state.

Input:

- action (Array of Numbers): the action proposed based on the current state (observation) of the environment.

Output:

- observation (Array of Numbers): the next state (next observation) resulting from applying the action on the current state.
- reward (Number): a measure of the action taken which tell whether the action done on the current state (observation) is good or bad.
- done (Boolean): indicates if an episode is done or not.
- info (dictionary): info that can be useful for debugging purposes.

$reset$: resets the environment and return it to the initial state.

Input:

- none.

Output:

- observation (Array of Numbers): the initial observation of the environment.

$render$: renders the current state of the environment.

Input:

- mode: sets the rendering mode of the environment.

Output:

- none.

$close$: terminate the environment instance and deletes it.

Input:

- none.

Output:

- none.

$apply\_blackouts$: changes the base stations electricity status according to the current step in the episode.

Input:

- time (Number): the current time of the episode.

Output:

- none.

set_action : sets an action on a specified base station.

Input:

- action (Number): the action index applied to base station.
- bs (Number): the base station whom the action is applied to.

Output:

- none.

$watt\_to\_dbm$: converts the power unit from watt to dbm (decibel-milliwatts).

Input:

- watt (Number): the power valued measured in watts.

Output:

- dbm (Number): the converted power measured in dbm (decibel-milliwatts).

dbm_to_watt : converts the power unit from dbm (decibel-milliwatts) to watt.

Input:

- dbm (Number): the power measured in dbm (decibel-milliwatts).

Output:

- watt (Number): the converted power valued measured in watts.

$path\_loss$: calculates the power loss of the signal transmitted from the sender (base station) to the receiver (mobile station).

Input:

- distance (Number): the distance between the sender (base station) and the receiver (mobile station) measured in meters.

Output:

- loss (Number): the expected power loss value measured in dbm.

$power\_grid$: calculates the power received at any point of the grid from any sender (base station) from RAN.

Input:

- bs_loc (Array of Coordinates): Positions of each base station in the RAN.
- bs_power (Array of Numbers): Initial power values of the base stations measured in dbm.
- points

Output:

- power_grid (Tensor of Numbers): has n_bs matrices and each matrix represents the power received from the base station at any point from the grid.

$get\_bit\_rate\_sinr$: calculates the bit rate using Shannon-Hartley equation.

Input:

- bandwidth (Number): the bandwidth used in the RAN for communication.
- signal (Matrix of Numbers): value of the signal power received at any point at the grid in watts.

- interference (Matrix of Numbers): value of the interference power received at any point at the grid in watts.

Output:

- bit_rate (Matrix of Numbers): the bit rate at any point at the grid in bits per second.

$get\_bit\_rate\_sinr$: calculates the bit rate from the power_grid tensor.

Input:

- power_grid (Tensor of Numbers): has n_bs matrices and each matrix represents the power received from the base station at any point from the grid.

Output:

- bit_rate (Matrix of Numbers): the bit rate at any point at the grid in bits per second.

get_state : returns the current state of the environment.

Input:

- none.

Output:

- state (Array of Numbers): the current state of the environment.

$baseline\_action$: applies the baseline action on a specified base station.

Input:

- bs_index (Number): the index of the base station the baseline action is applied to.

Output:

- none.

$bulk\_actions$: applies a set of actions on all of the base stations on the RAN each with its respective action.

Input:

- actions (Array of Numbers): the actions to be applied on the base stations, each action has to be applied on its corresponding base station.

Output:

- none.

$apply\_baseline$: applies the baseline action on all of the base stations on the RAN.

Input:

- none.

Output:

- none.

$bit\_rate\_cost\_function$: calculates the reward from the bit rate grid.

Input:

- b_rate (Matrix of Number): the bit rate grid of the system.

Output:

- reward (Number): the calculated reward of the bit rate grid.

get_reward : calculates the reward of the current action-state pair.

Input:

- none.

Output:

- reward (Number): the calculated reward of the current action-state pair.

end class

This block represents a rough implementation of the RAN environment block as a standardized open-ai gym environment. Which leaves a lot of scalability and flexibility to simulate various types of RAN networks and integrate them into different agents including deep learning and deep reinforcement learning agents. The state (observation) in this environment is an array of booleans represents the electricity condition of the base stations in RAN (e.g. 1 is for blackout, 0 is for not-in-blackout).
The figure 3.3 shows how the $step$ function is implemented in the RAN environment class. It accepts the agent desired actions as its input applies them to the environment calculates the next state (observation), reward and whether the episode is done and returns the result of those calculation back to the agent. These actions in this case is an array of binary numbers which tells the environment to shut down in case of zero or start up the corresponding base station if the value is

Algorithm 11 env.step

1: **Input:** $actions$ an array of the actions taken to each base station
2: **Output:** $observation$ next state, $reward$ number, $done$ boolean
3: Execute actions given by agent through env.bulk_actions.
4: Calculate the reward given by the current state and action pair using env.get_reward and assign $reward$ the value returned by the function.
5: Increment the timer of the environment by $step\_size$.
6: Check if the electricity state of the base stations changes as the time progresses and if true update it.
7: Check if the timer value exceeds the episode length and set $done$ by true or false depending on the answer of the question.
8: Through env.get_state obtain the next state and store it in $observation$.
9: return $observation$, $reward$, $done$.

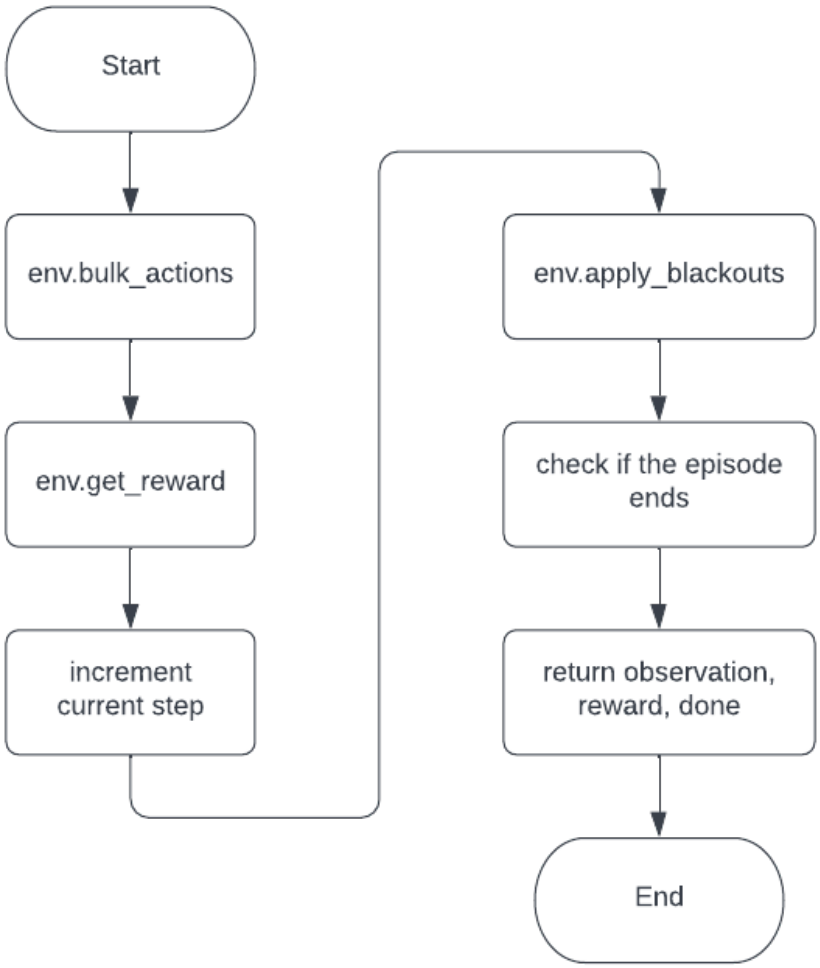


Figure 3.3: RAN Environment step

3.3.1 Reward Engineering

AI agents are becoming far more broad and self-contained. A "Reward Engineering Principle" has to be established as reinforcement learning based AI systems get more broad and autonomous, designing reward mechanisms to elicit desirable behavior becomes both more crucial and more complex. While early AI research may have overlooked reward design in favor of focusing solely on the problems of efficient, flexible, and effective achievement of arbitrary goals in a variety of environments, the reward engineering principle will have a medium and long-term impact on modern AI research, both theoretical and applied. So as to have a flexible and effective reward that can encapsulate both short term and long term rewards, the following reward rules are proposed, those rules are:

- There is a constant reward that is always added at each step, in this case a positive small number would be perfect
- min_cost which is the minimum speed across the grid. As the performance of RAN increases it also increases, so it is intuitive to add it.
- $disparity$ it is the ratio of the difference between the maximum speed and the minimum speed across the grid compared to maximum speed across the grid. This reward is reversely proportional to the RAN total performance.
- br_cost which is equal to the percentage of the grid which have less bit rate than a certain threshold. The threshold in this case equals 1MB/sec. This reward is reversely proportional to the RAN total performance.
- $elec\_cost$ is the cost of running all base stations in blackouts, it is assumed to be 10 watts (transmission power). This reward is reversely proportional to the RAN total performance.

Combining all these types of rewards and punishments gives rise to the final equation used as the reward function for this environment as shown in equation 3.11. The lower limit of this reward function happens when no base station is working as min_cost equals 0 and br_cost equals 10, although $elec\_cost$ is 0, most part of the punishment comes from $disparity$ which is supposedly tends to go to infinity as it is described in equation 3.11 but its valued is capped at 1000 to not cause training issues when using deep learning models. All in all if those values are plugged in the equation 3.11 the lower limit should equal -990. Determining the upper limit is quite hard task and is network dependant but it is guaranteed that it is less than or equal 20 as min_cost can not overwhelm the other terms in the equation 3.11.

$$
\begin{aligned}
reward &= c + min\_cost - br\_cost - disparity - elec\_cost \\
c &= 20 \\
min\_cost &= min(bit\_rate\_grid) \\
disparity &= \frac{max\_speed - min\_speed}{max\_speed} \\
elec\_cost &= running\_blackout\_bs * 10 \\
br\_cost &= \frac{(\%bit\_rate\_grid < 1MB) * 10}{100}
\end{aligned}
\tag{3.11}
$$

Algorithm 12 env.get_reward

1: **Input:** none
2: **Output:** $reward$ number: indicator of the performance of the agent in the latest action taken.
3: Obtain the state after executing the actions.
4: Use env.grid_height and grid_width to generate enough points to approximate the overall performance of the RAN.
5: **for** point in grid **do** Calculate the distances between the point and all base stations.
6: Calculate all signal received at point from all operating base stations via env.path_loss.
7: Obtain the maximum signal value and treat other signal values as interference.
8: Use Shannon-Hartley formula to calculate bit rate at the point.
9: Append bit rate value to the grid.
10: **end for**
11: Obtain the minimum bit rate value in MB/sec.
12: Obtain the maximum bit rate value in MB/sec.
13: Apply disparity formula proposed in this research.
14: Obtain the cost of operating base stations in blackout.
15: Sum all values together and store the value in $reward$.
16: return $reward$.

3.3.2 Baseline Model

A simple baseline model is implemented in the environment in a built-in fashion. This baseline model is used later as a benchmark to assess the performance of other models trained on this environment. Baseline model focuses on the concept of using as little power in blackouts as possible. This model stops base stations which currently in blackout and off-grid and operates those with accessibility to electricity.

Algorithm 13 env.baseline_actions

```
1: Input: none.
2: Output: none.
3: Obtain the current electricity state of each base station.
4:
5: for base station in RAN system
6:  do
7:     if base station is currently not in blackout then
8:         turn-on the base station
9:     else if base station is currently in blackout then
10:        turn-off the base station
11:    end if
12: end for
```

3.4 Experiment

The built-in baseline model in the environment is used first to generate a benchmark for rewards to compare the models trained later on. The models used are implemented in Stable-Baselines3.The open-source framework Stable-Baselines3 (SB3)[35] implements seven widely used model-free deep Reinforcement Learning techniques. To obtain high-quality implementations that match previous outcomes, It take great effort to follow software engineering best practices. Each approach has been tested in a variety of scenarios and compared to previous implementations. Its implementation covers 95% of the code, and the active user base evaluating modifications guarantees that any implementation problems are kept to a minimum.Features offered by Stable-Baselines3:

- API that is easy to use. It just takes a few lines of code to train an agent in Stable-Baselines3, after which the agent may be asked for actions. This makes it simple for researchers to employ the standard algorithms and components in their investigations.
- Documentation. SB3 comes with a lot of code API documentation. It also contains a user guide with a set of realistic examples that covers both basic and advanced usage.
- Implementations of the highest caliber. The agent learning curves are compared to published findings to verify algorithms. Furthermore, all functions are typed (parameter and return types) and described in a consistent manner, with unit tests covering the majority of them. Continuous integration validates the code style and documentation, as well as ensuring that all changes pass unit tests and type checks.
- Comprehensive. The following state-of-the-art on- and off-policy algorithms, often used as experimental baselines, are included in Stable-Baselines3: A2C[36],

PPO[20], DDPG[37], SAC[38], TD3[39], HER[40], and DQN[2] are some of the models that have been implemented.

In this thesis Stable-Baselines3 models A2C and PPO are used as the models for the experiment. The experiment includes several episodes of 24 hours and a step size of 1.5 hours per step. Those models are tested against simple baseline policy which is shutting down all base stations that the grid electricity does not reach them at that point on time and running any other base station that has access to the grid electricity.

3.4.1 Experiment Parameters

The simulation settings used to test the proposed approach are listed in table 3.1. Users are assumed to be evenly dispersed throughout the RAN deployment region. The number of base stations in the system is 16 with average distance between nearest neighbors of 500 meters. the base stations are positioned in a grid where the mean distance between adjacent columns and adjacent rows is 500 meters with an offset of ± 50 meters. Also any base station in RAN is assumed to be operating with a single antennae on the same frequency with a bandwidth of 10 Mega-Hertz[33].

Table 3.1: Simulation Parameters

Parameter	**Value**
Base Stations	16
Area	6.25 km^2
Blackout Slots	3
Avg Blackout Time	4 hours
User Distribution	Normal
Transmitted Power	40 dbm
Bandwidth	10 MHz
Cellular Standard	LTE
Frequency	2.6 GHz
Total Time	24 hours
Step Size	1.5 hours

Locations of the base stations in the RAN (Radio Access Network) is synthesized by a random function (with a fixed seed to obtain the same values for replication purposes) by rules stated above, the base stations are placed in a $4x4$ grid with a total of 16 base stations. The mean distance between each adjacent rows or adjacent columns is 600

meters with an offset of ± 60 meters to simulate irregularities that may happen when placing base stations. Table 3.2 and figure 3.4 shows the grid placement of the base stations and the exact location of them.

Table 3.2: Base Stations Locations

(2389,577)	(2359,1241)	(2382,1756)	(2454,2380)
(1768,644)	(1782,1225)	(1792,1746)	(1819,2356)
(1225,599)	(1160,1236)	(1255,1848)	(1248,2349)
(557,590)	(556,1152)	(605,1748)	(560,2374)

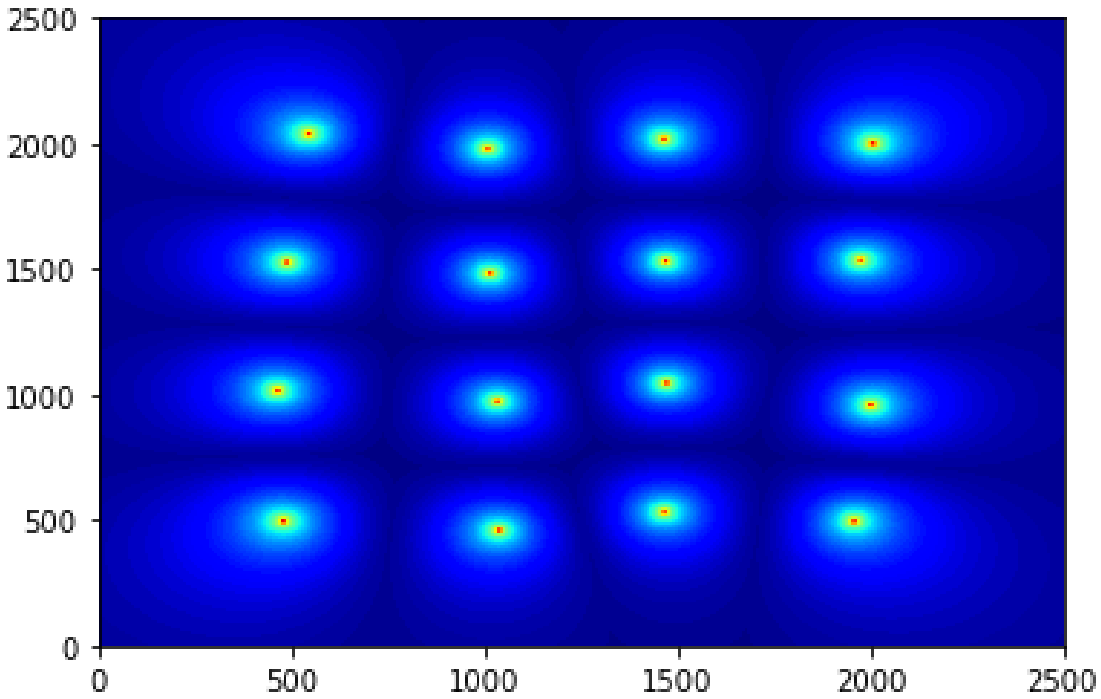


Figure 3.4: Locations of Base Stations in Grid, as shown in red

To simulate the effects of blackouts on the RAN first a blackout schedule is first generated. The method used to synthesize blackout schedule is by first determining the slots of the blackout as prescribed by table 3.3 there 3 slots for blackouts each one of them lasts for 4 hours and 2 slots are overlapping. Table 3.4 shows each base station in the grid and their slot in the blackout schedule.

Table 3.3: Blackouts Schedules Slots

Time in seconds	**Time in hours**
(10800,25200)	(03:00,07:00)
(21600,36000)	(06:00,10:00)
(50400,64800)	(14:00,18:00)

Table 3.4: Base Stations Blackouts

(21600,36000)	(50400,64800)	(10800,25200)	(10800,25200)
(50400,64800)	(50400,64800)	(21600,36000)	(50400,64800)
(21600,36000)	(50400,64800)	(10800,25200)	(10800,25200)
(10800,25200)	(10800,25200)	(21600,36000)	(50400,64800)

Chapter Four

Results

4.1 Training

both the PPO and A2C models have been trained on Intel Core i7-2620M Processor 2.7GHz 6GB DDR3 RAM computer on three passes

As the training step progressed we came out to realize several facts about both models:

- PPO model performs better than A2C model provided enough training time for both models
- A2C model has faster convergence steps than PPO model
- PPO model is much more stable than A2C model and does not suffer from rebounds as for A2C model
- PPO model can explore environment state and actions better than A2C model
- PPO model can exploit with much greater effect the explored state-action dues better than A2C model

4.1.1 A2C Training

In this thesis A2C model (Advantage Actor-Critic) is used as one of the 2 models other than the baseline as it is one of the state-of-art reinforcement learning models used currently in research along with PPO (Proximal Policy Optimization). The results are then compared.

One of the main advantages of A2C over its predecessors is that it has faster convergence with more stability in performance, after training it in 3 passes as in figures (4.1, 4.2, 4.3) with each pass taking up to 12 hours of training and a total of 36 hours of training as shown in figure 4.4 which represents the training progression as the rewards evolves during training. (Note that as the lower bound of rewards is -990 which removes details from plots every -990 was replaced with -80 for visibility concerns)

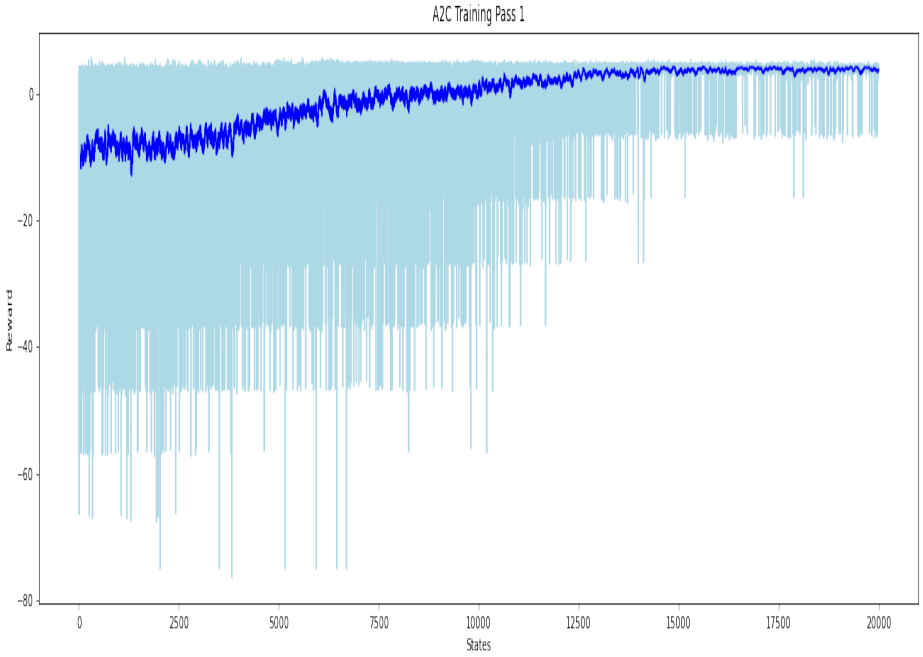


Figure 4.1: First phase of training A2C Model

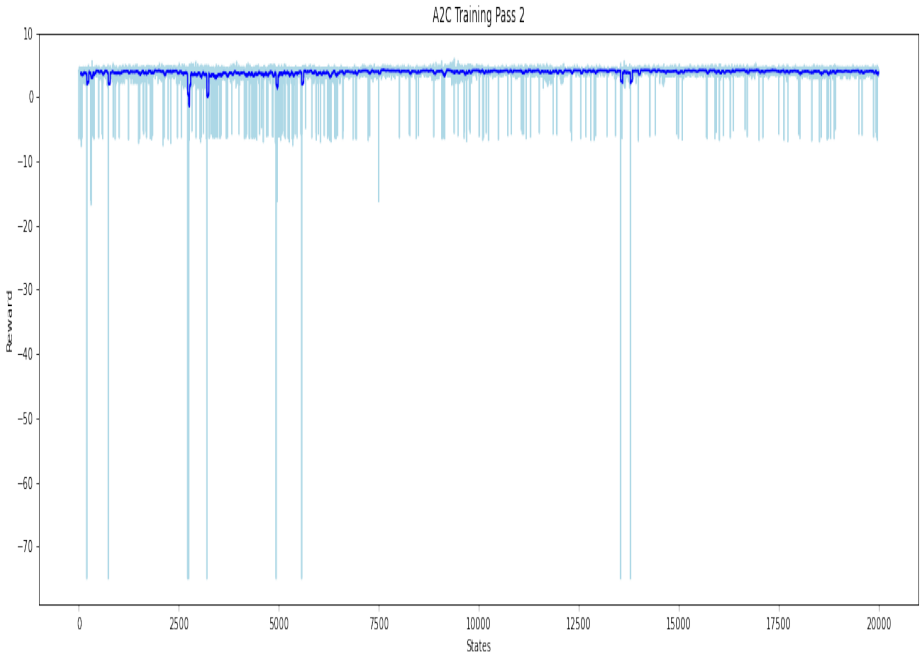


Figure 4.2: Second phase of training A2C Model

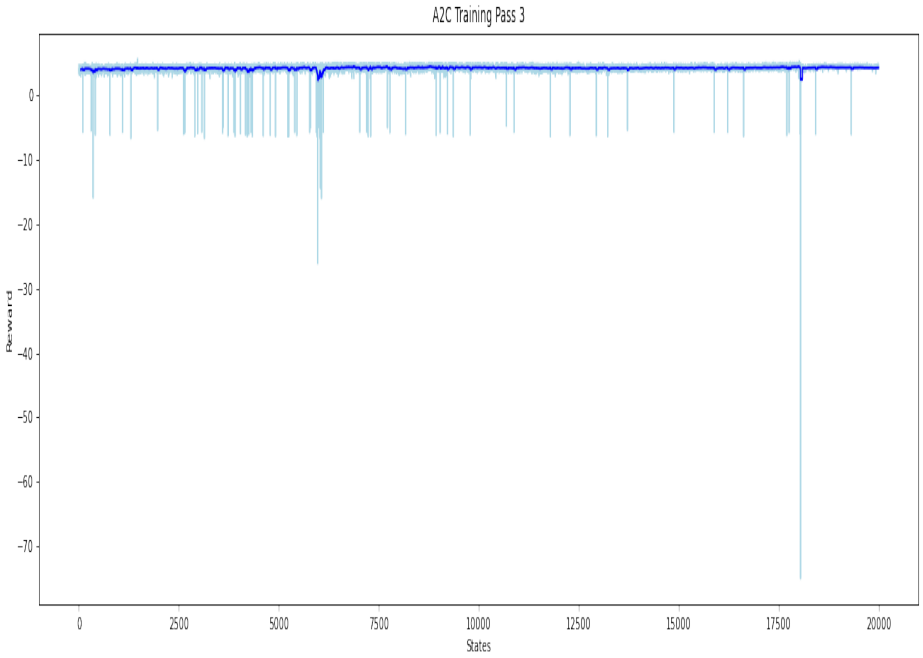


Figure 4.3: Third phase of training A2C Model

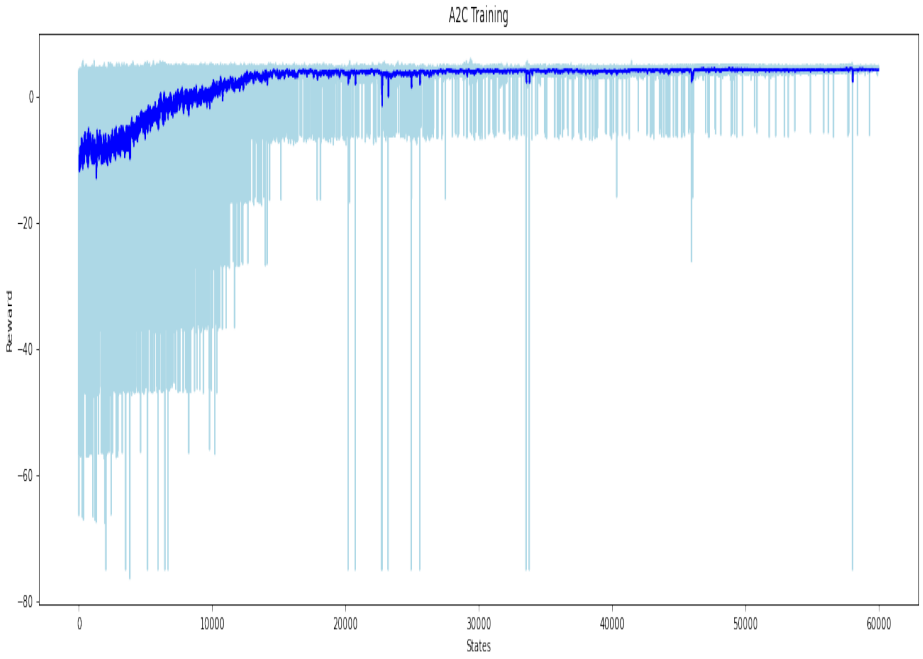


Figure 4.4: The whole training process of A2C Model

4.1.2 PPO Training

The second model used in this thesis is PPO (Proximal Policy Optimization) which came after A2C as to fix the stability issues found in A2C. It provide high stability in performance.

One of the main advantages of PPO over its predecessors is that it is high stability in performance although it may require more time to converge, after training it in 3 passes as in figures (4.5, 4.6, 4.7) with each pass taking up to 12 hours of training and a total of 36 hours of training as shown in figure 4.8 which represents the training progression as the rewards evolves during training. (Note that as the lower bound of rewards is -990 which removes details from plots every -990 was replaced with -80 for visibility concerns)

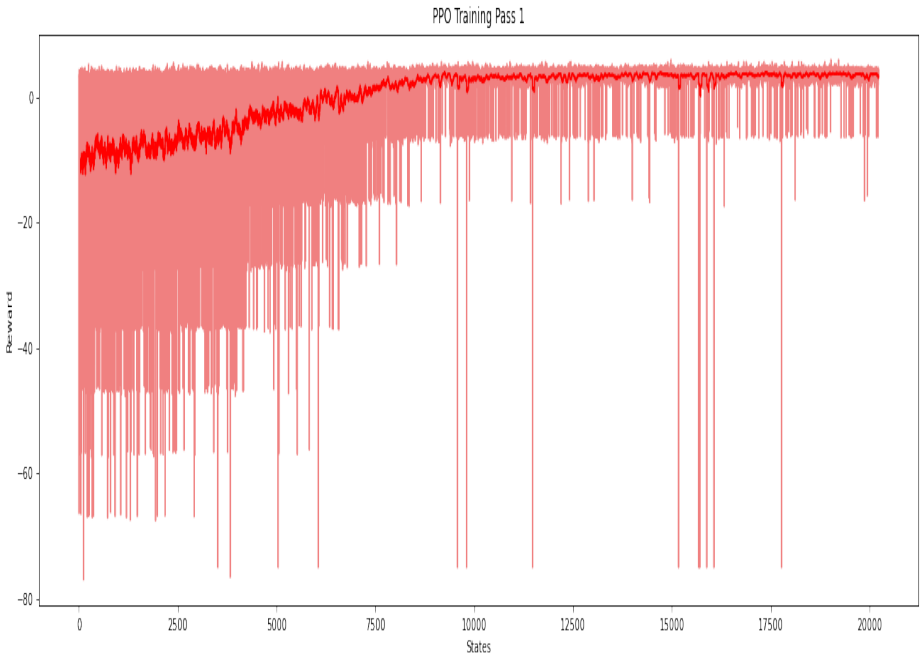


Figure 4.5: First phase of training PPO Model

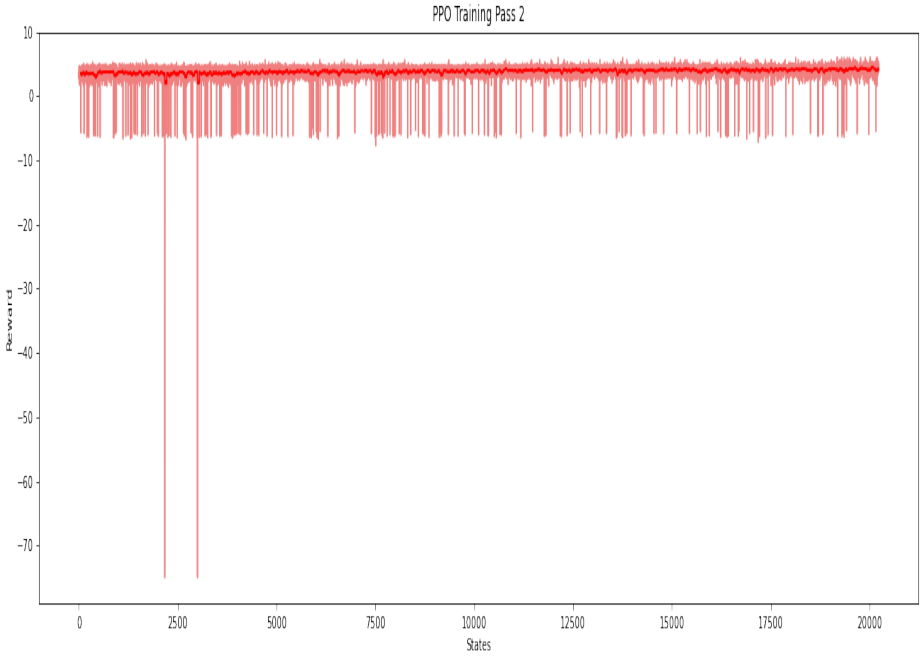


Figure 4.6: Second phase of training PPO Model

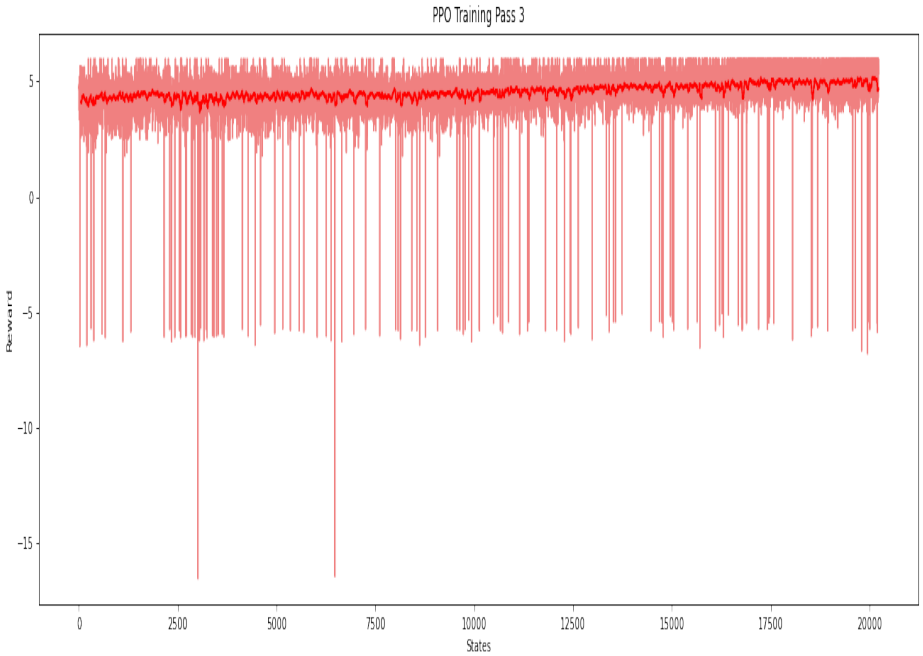


Figure 4.7: Third phase of training PPO Model

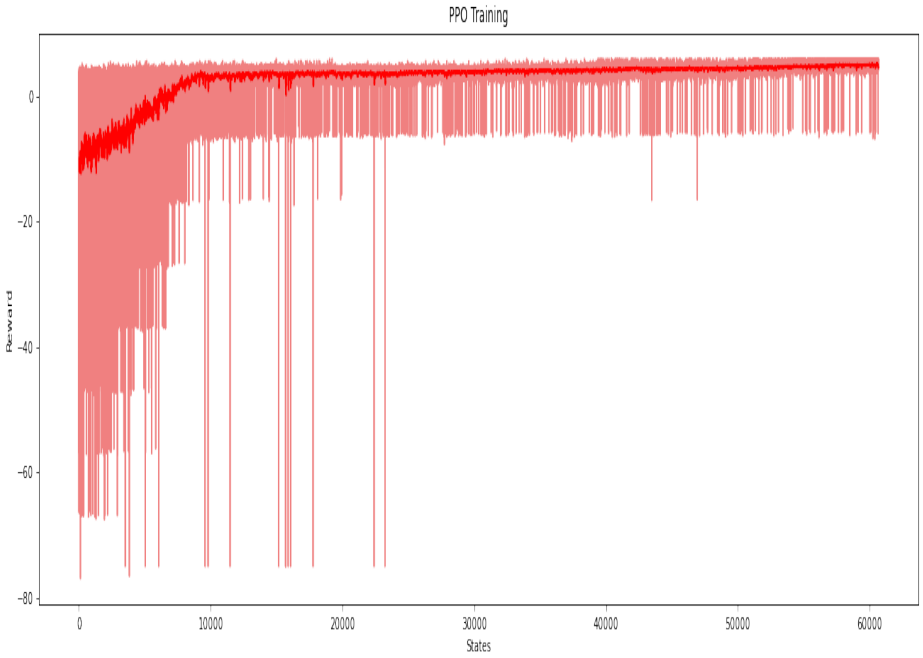


Figure 4.8: The whole training process of PPO Model

4.2 Exploration Vs. Exploitation

The exploration-exploitation trade-off is a well-known issue that arises when a learning system is forced to make several decisions with unclear outcomes. In essence, a decision-making system with limited information of the environment faces a dilemma of whether to repeat actions that have worked successfully in the past (exploit) or to create new options in the hopes of gaining even higher benefits (explore).

Here we compare how A2C and PPO models explore the environment as shown in figures 4.9 and 4.10 respectively as what is not explored can not be exploited and as shown in the figures the conclusion of PPO having better exploration schema than A2C model and with a fine margin as the model explored better solutions in most cases and the rest found equal solutions to A2C solutions.

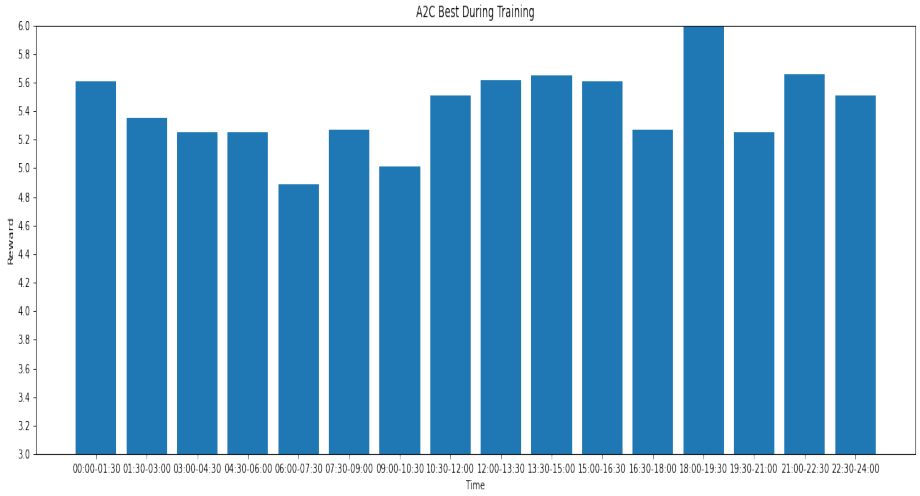


Figure 4.9: Best reward values explored by A2C model during training

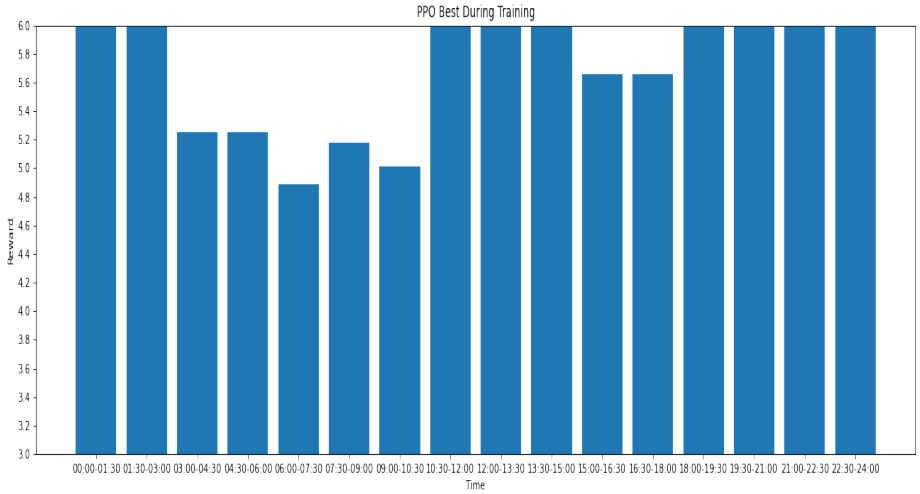
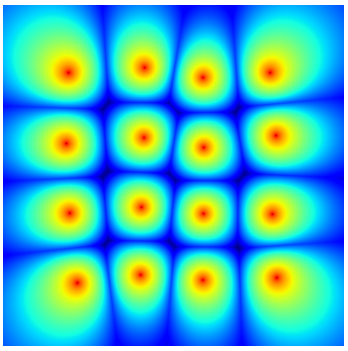


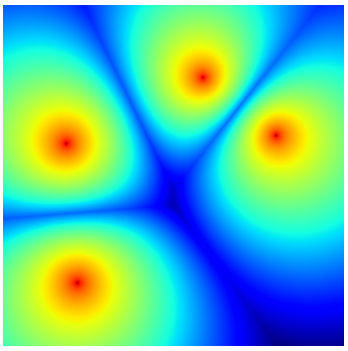
Figure 4.10: Best reward values explored by PPO model during training

4.3 Recommended Actions

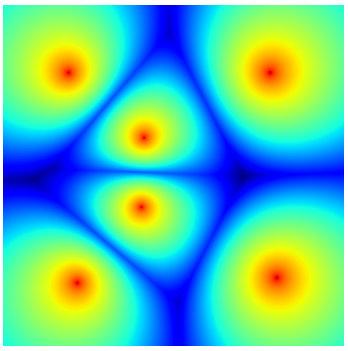
As one episode of training have 16 steps in total, after training both models we compared their final solution of each state against the baseline solutions, in the following figures the bit rate grid along with the active base stations are shown for the models and baseline in each state. (note that the values in the plot are log of the real values calculated)



(a) Baseline after state 1

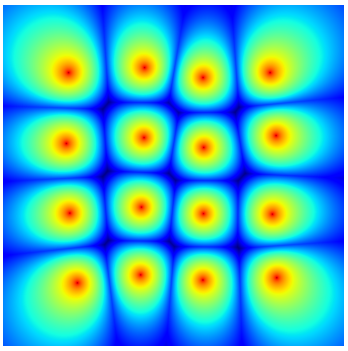


(b) A2C state after state 1

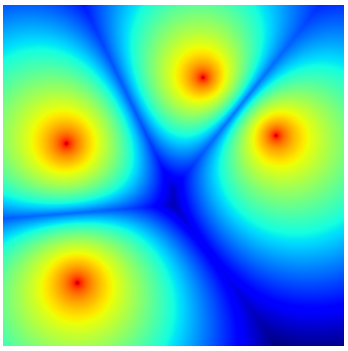


(c) PPO state after state 1

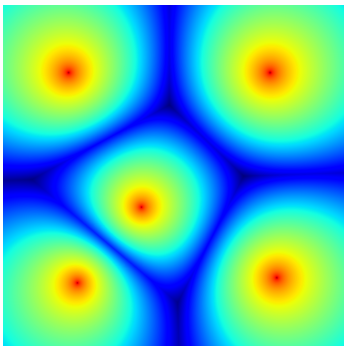
Figure 4.11: Bit rate distribution in scenario 1



(a) Baseline after state 2

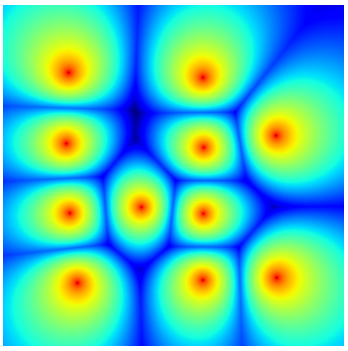


(b) A2C state after state 2

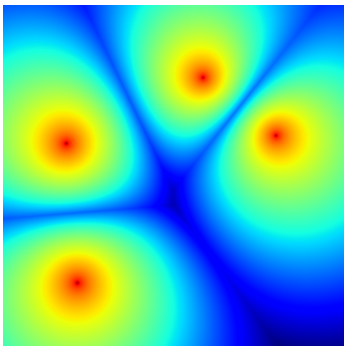


(c) PPO state after state 2

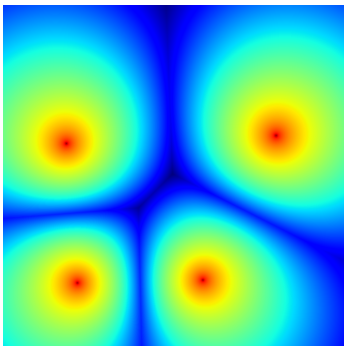
Figure 4.12: Bit rate distribution in scenario 2



(a) Baseline after state 3

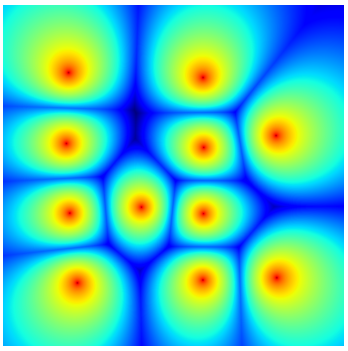


(b) A2C state after state 3

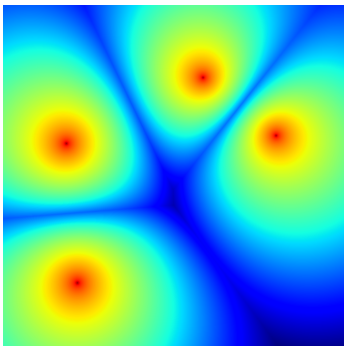


(c) PPO state after state 3

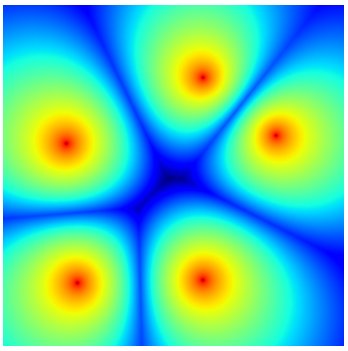
Figure 4.13: Bit rate distribution in scenario 3



(a) Baseline after state 4

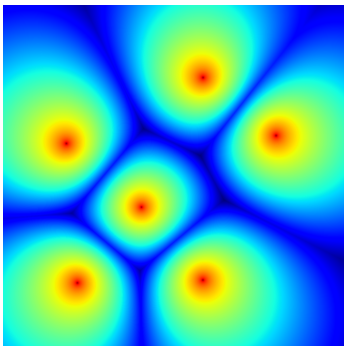


(b) A2C state after state 4

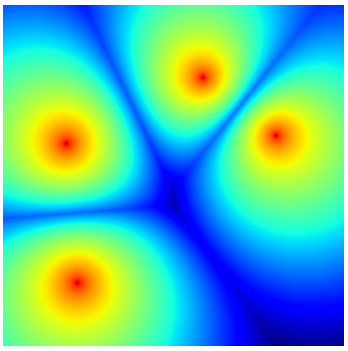


(c) PPO state after state 4

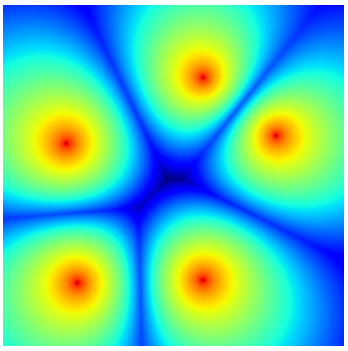
Figure 4.14: Bit rate distribution in scenario 4



(a) Baseline after state 5

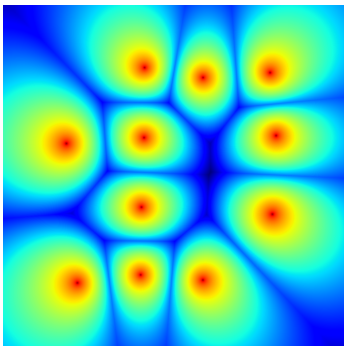


(b) A2C state after state 5

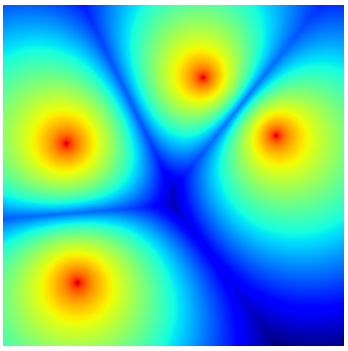


(c) PPO state after state 5

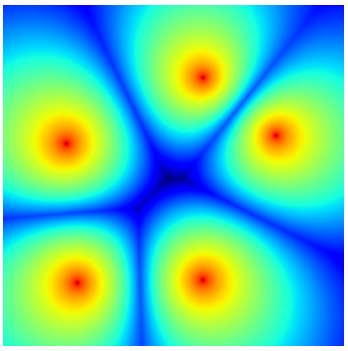
Figure 4.15: Bit rate distribution in scenario 5



(a) Baseline after state 6

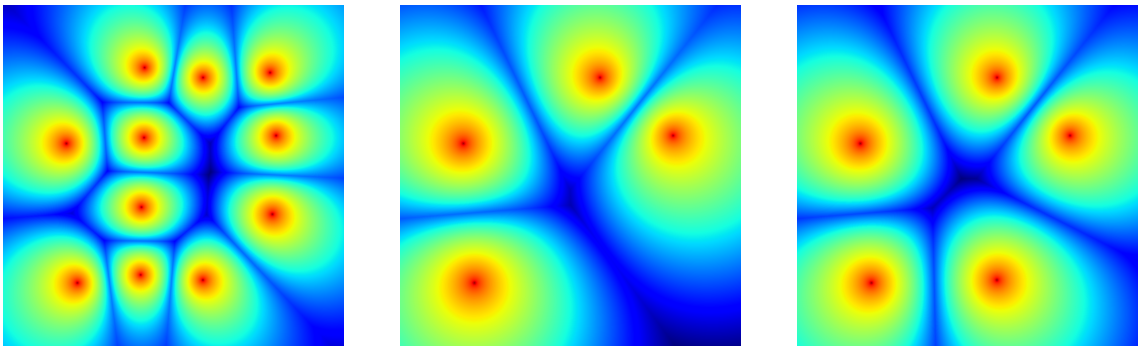


(b) A2C state after state 6



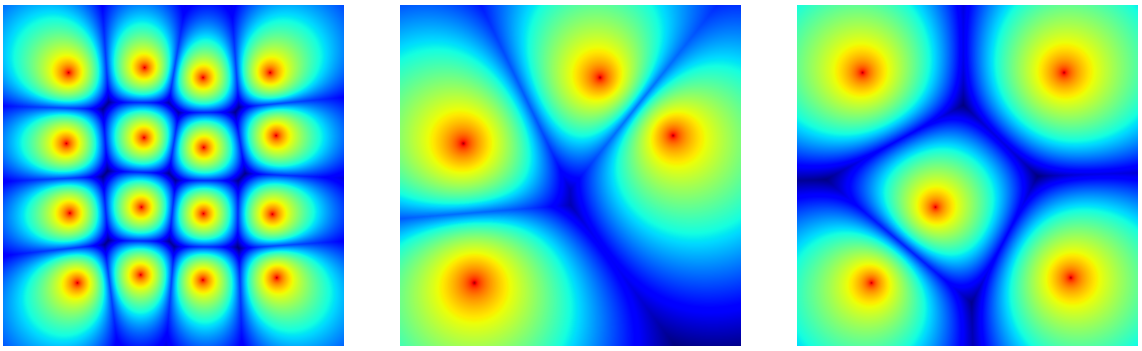
(c) PPO state after state 6

Figure 4.16: Bit rate distribution in scenario 6



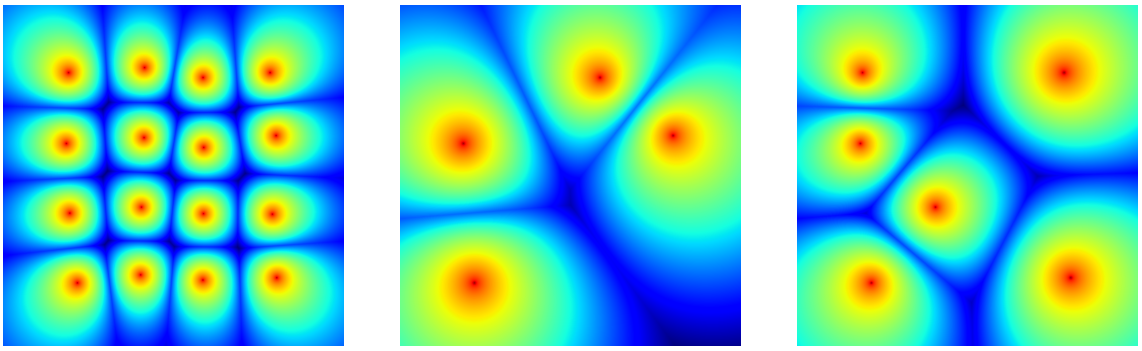
(a) Baseline after state 7 (b) A2C state after state 7 (c) PPO state after state 7

Figure 4.17: Bit rate distribution in scenario 7



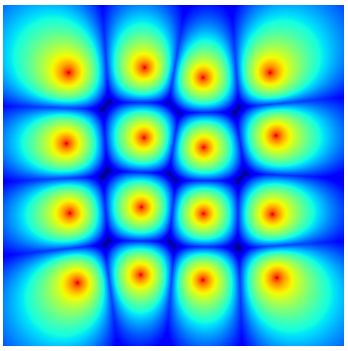
(a) Baseline after state 8 (b) A2C state after state 8 (c) PPO state after state 8

Figure 4.18: Bit rate distribution in scenario 8

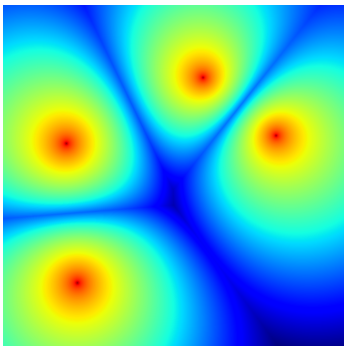


(a) Baseline after state 9 (b) A2C state after state 9 (c) PPO state after state 9

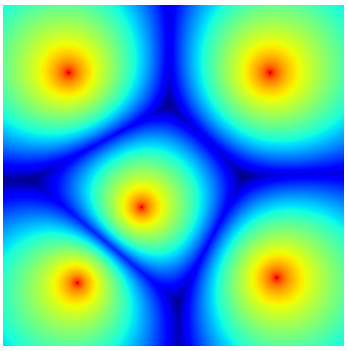
Figure 4.19: Bit rate distribution in scenario 9



(a) Baseline after state 10

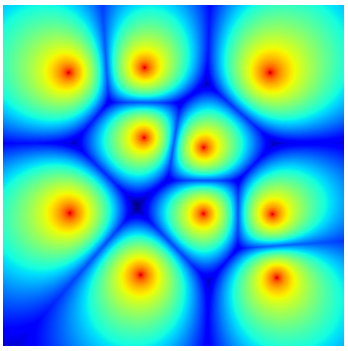


(b) A2C state after state 10

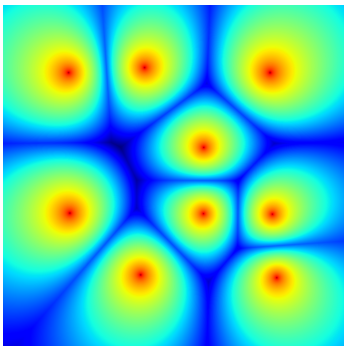


(c) PPO state after state 10

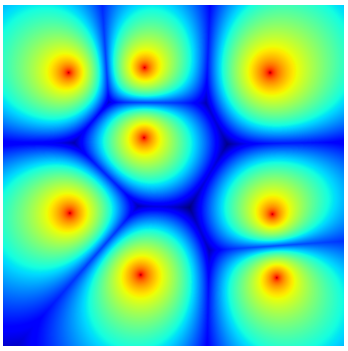
Figure 4.20: Bit rate distribution in scenario 10



(a) Baseline after state 11

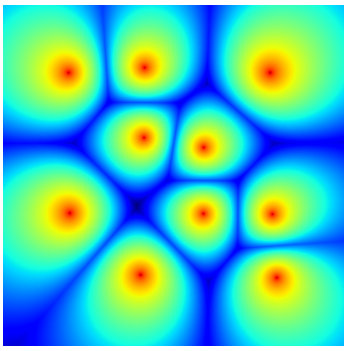


(b) A2C state after state 11

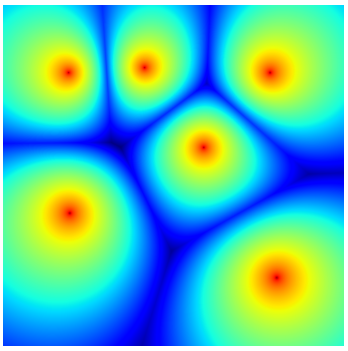


(c) PPO state after state 11

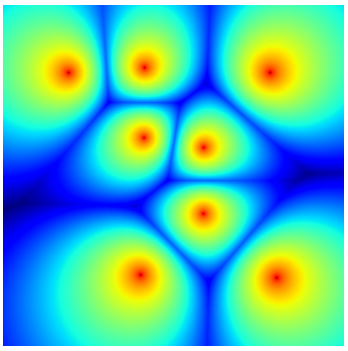
Figure 4.21: Bit rate distribution in scenario 11



(a) Baseline after state 12

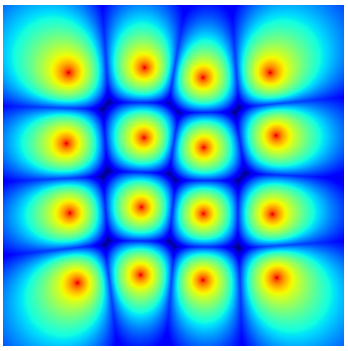


(b) A2C state after state 12

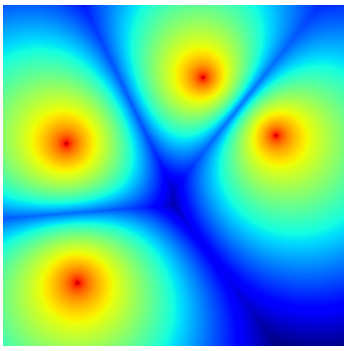


(c) PPO state after state 12

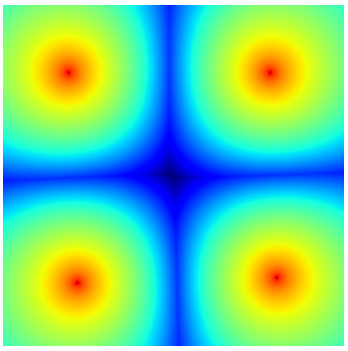
Figure 4.22: Bit rate distribution in scenario 12



(a) Baseline after state 13

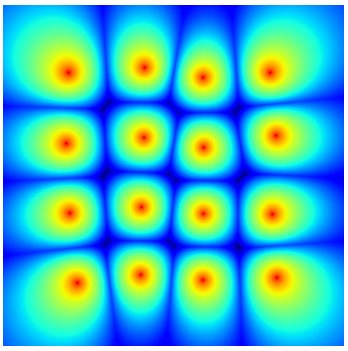


(b) A2C state after state 13

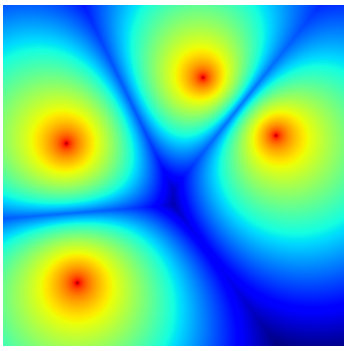


(c) PPO state after state 13

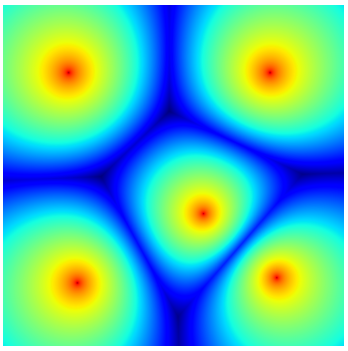
Figure 4.23: Bit rate distribution in scenario 13



(a) Baseline after state 14

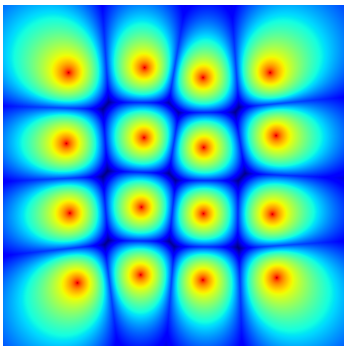


(b) A2C state after state 14

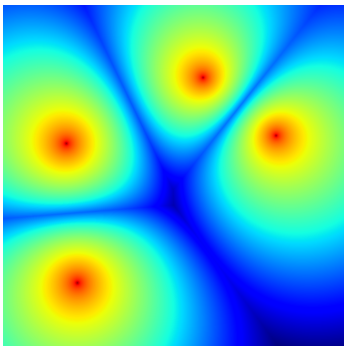


(c) PPO state after state 14

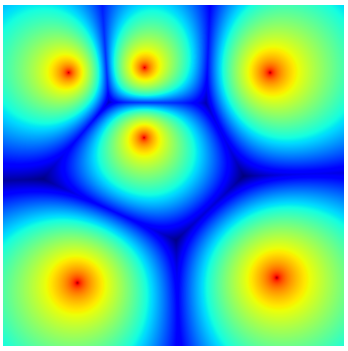
Figure 4.24: Bit rate distribution in scenario 14



(a) Baseline after state 15

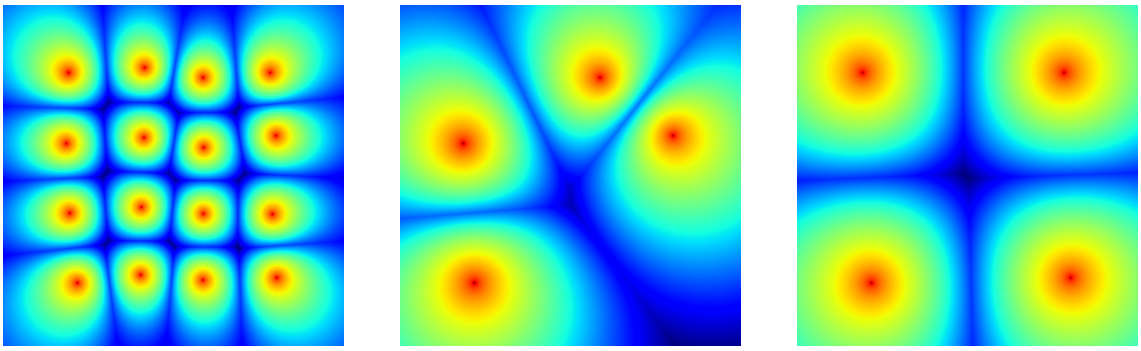


(b) A2C state after state 15



(c) PPO state after state 15

Figure 4.25: Bit rate distribution in scenario 15



(a) Baseline after state 16 (b) A2C state after state 16 (c) PPO state after state 16

Figure 4.26: Bit rate distribution in scenario 16

As shown in figure 4.27, the recommend actions rewards suggested by the baseline model has a lower bound of 3.2 and a higher bound of 4.4 and does a fairly good job of maintaining quality of service. But the recommended actions of both A2C and PPO - as shown in figures 4.28 and 4.29 respectively - surpasses that of the baseline the lower bounds have increased dramatically for both models compared to the baseline with the PPO have very high upper bound compared to both A2C and the baseline thus declaring PPO as the undisputed best out of the three approaches.

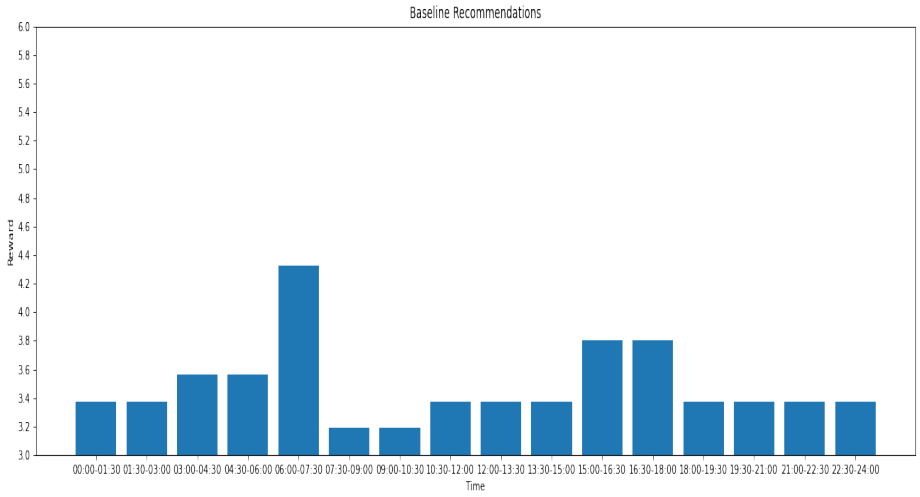


Figure 4.27: Rewards for the recommended actions of the baseline model

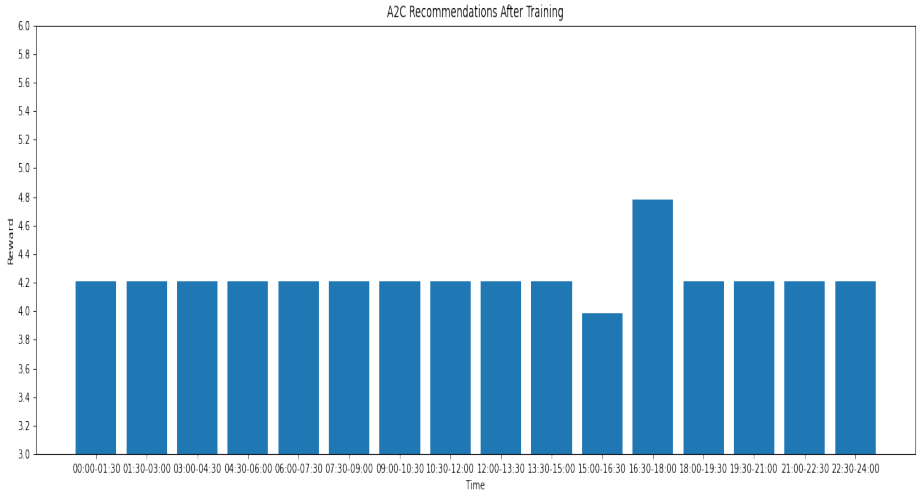


Figure 4.28: Rewards for the recommended actions of A2C model

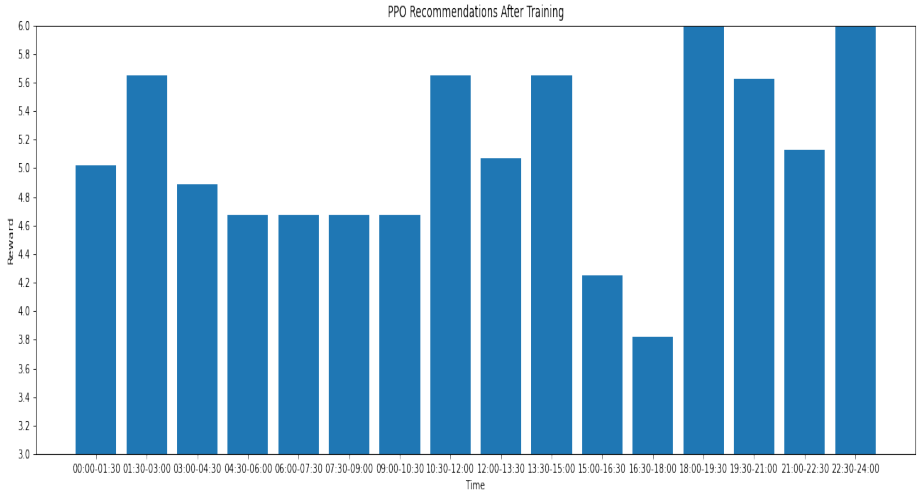


Figure 4.29: Rewards for the recommended actions of PPO model

Chapter Five

Review and Conclusion

5.1 Conclusion

In this thesis, the adequacy of available environments in replicating the RAN network model was analysed and investigated, several network simulators were studied in the process, but all of them fall short when replicating load-shedding effects and integrating with Python and deep learning libraries, drawbacks that severely hinder the goal of this thesis. Which only leaves the option of implementing a new standardized simulator. The RAN system model behind the proposed methodology is presented and discussed. The proposed methodology is based on characterizing the throughput within the RAN.

A power propagation model was introduced depicting the propagation loss parameters with regard to base stations in urban areas and formulating the relationship between propagation power loss and distances between transmitters and receivers of a network.

The thesis main in concern QoS measurement was channel capacity, which was calculated using Shannon-Hartley theorem given the normalized channel capacity and the bandwidth of the base stations in the RAN.

Assumptions about network environment and electrical supply are made before specifying the parameters and circumstances assumed for the suggested methodology's procedure. A significant assumption made is that RAN's operation area is thought to be in a densely populated region, another assumption is that all base stations in RAN are connected to a global unstable electricity grid that is moderated to cut-off some base stations electrical supply in a pre-determined schedule.

In order to implement the environment needed to simulate any RAN network with the required integration to copulate the environment with any deep reinforcement learning agent or any kind of agents. The standardized environment implementation (OpenAI gym) is used to resolve two problems that hampers RL research (The lack of consistency in the contexts utilized in publications and the need of better benchmarks). The implemented environment took into consideration network scalability and flexibility to simulate various types of RAN networks.

Relevant parameters of grid, base stations, blackouts and timing were inspected and later selected with regard to their significance to the simulation concerns. The simulation configurations used to test the proposed approach are listed in table 3.1.
Reward mechanisms were designed to elicit flexibility and effectiveness of RAN networks. Multiple reward rules are proposed, that include minimum bit rate offered by RAN system, disparity in speed, and cost of operating base stations during blackout which are later combined to formulate a final equation that defines the reward function.
An environment that can simulate RAN (Radio Access Network) with both integration with python and the ability to simulate load-shedding is implemented with taking into consideration scalability and flexibility, this environment calculates both quality of service KPIs (Key Performance Indicator) i.e. bit rate and power usage on-grid and off-grid. This environment is then used to train 2 reinforcement learning models A2C (Advantage Actor-Critic), and PPO (Proximal Policy Optimization). These models are then compared to an built-in baseline in the environment. The baseline model stops base stations which currently can not access electricity and operates those that can access it. The two models used surpassed the baseline in performance although PPO surpasses A2C in performance.
Although the operation modes proposed by both PPO and A2C have much less operating base stations than what is proposed by the baseline model, they generally achieve better performance than the baseline. This may be due to the fact that less base stations means less interference and these two models learnt how to decrease interference levels to its minimum.

5.2 Research Limitation

When applying all algorithms in the proposed environment, there was some limitation we faced on our project which were that:

- Domain knowledge in this field is very constricted specially in Sudan by the competitive nature of the Large tech companies operating.
- Each step consumes time to execute on an average computer, which slows down the learning process by a considerable factor.
- Only two Reinforcement Learning algorithms were used, one is an Actor-Critic algorithm, and the other is a Policy-Gradient method.
- Little to no documentation is found for implementing multi-agent environments in OpenAI Gym, so instead a single agent environment was implemented.

5.2.1 Reward Engineering

Another way to achieve better results and better used the explored states in the proposed environment is by changing how to deal with reward to get better results which can be achieved by:

- Increase the number of episodes or steps per episode, which in turn increase training time.
- Search for the best hyper-parameters for each algorithm taking, but hyper-parameters search is time consuming and bound by trail and error.
- Add layer normalization, Layer normalization add stability to the learning process in wide range of reward scaling.

5.3 Future Work

- Understanding power consumption of base station to better incorporate it into the rewarding function of the environment than what is implemented in the environment.
- Using decentralized method of controlling the base stations rather than centralized method proposed in this thesis, which allows to use multi-agent reinforcement learning methods and achieve better scalability in RAN.
- Implementing this environment to support both GPUs (Graphics Processing Unit) and TPUs (Tensor Processing Unit) in order to increase the speed of processing a step of the episode which will dramatically increase the learning process.

References

[1] R. S. Sutton, "Introduction: The challenge of reinforcement learning," pp. 1–3, 1992.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[3] M. C. Huebscher and J. A. McCann, "A survey of autonomic computing—degrees, models, and applications," *ACM Computing Surveys (CSUR)*, vol. 40, no. 3, pp. 1–28, 2008.

[4] F. Hussain, S. A. Hassan, R. Hussain, and E. Hossain, "Machine learning for resource management in cellular and iot networks: Potentials, current solutions, and open challenges," *IEEE communications surveys & tutorials*, vol. 22, no. 2, pp. 1251–1275, 2020.

[5] X. Li, J. Fang, W. Cheng, H. Duan, Z. Chen, and H. Li, "Intelligent power control for spectrum sharing in cognitive radios: A deep reinforcement learning approach," *IEEE access*, vol. 6, pp. 25 463–25 473, 2018.

[6] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[7] O. Simeone, "A very brief introduction to machine learning with applications to communication systems," *IEEE Transactions on Cognitive Communications and Networking*, vol. 4, no. 4, pp. 648–664, 2018.

[8] S. Schmid, M. Sifalakis, and D. Hutchison, "Towards autonomic networks," in *IFIP TC6 International Conference on Autonomic Networking*, Springer, 2006, pp. 1–11.

[9] D. F. Bantz, C. Bisdikian, D. Challener, *et al.*, "Autonomic personal computing," *IBM Systems Journal*, vol. 42, no. 1, pp. 165–176, 2003.

[10] A. Computing *et al.*, "An architectural blueprint for autonomic computing," *IBM White Paper*, vol. 31, no. 2006, pp. 1–6, 2006.

[11] P. Arcaini, E. Riccobene, and P. Scandurra, "Modeling and analyzing mape-k feedback loops for self-adaptation," in *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, IEEE, 2015, pp. 13–23.

[12] T. A. Nguyen, M. Aiello, T. Yonezawa, and K. Tei, "A self-healing framework for online sensor data," in *2015 IEEE International Conference on Autonomic Computing*, IEEE, 2015, pp. 295–300.

[13] G. Auer, V. Giannini, C. Desset, *et al.*, "How much energy is needed to run a wireless network?" *IEEE wireless communications*, vol. 18, no. 5, pp. 40–49, 2011.

[14] B. Debaillie, A. Giry, M. J. Gonzalez, *et al.*, "Opportunities for energy savings in pico/femto-cell base-stations," in *2011 Future Network & Mobile Summit*, IEEE, 2011, pp. 1–8.

[15] A. G. Barto and R. L. AG, "Reinforcement learning: An introduction to," *A Bradford Book*, 1998.

[16] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[17] A. Choudhary, "A hands-on introduction to deep q-learning using openai gym in python," *Retrived from https://www. analyticsvidhya. com/blog/2019/04/introduction-deep-q-learningpython*, 2019.

[18] J. Peters and J. A. Bagnell, "Policy gradient methods.," *Scholarpedia*, vol. 5, no. 11, p. 3698, 2010.

[19] R. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with imperfect value function," *Advances in Neural Information Processing Systems*, vol. 12, pp. 1057–1063, 2000.

[20] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[21] S. Karagiannakos, "The idea behind actor-critics and how a2c and a3c improve them," *Sensors*, 2018.

[22] C. Wang and K. Ross, "On the convergence of the monte carlo exploring starts algorithm for reinforcement learning," *arXiv preprint arXiv:2002.03585*, 2020.

[23] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.

[24] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[25] B. B. Alexander Zai, *Deep Reinforcement Learning In Action*. Manning, 2019.

[26] M. Ahmed Mohamed Ahmed, *Traffic model based energy efficient radio access network*, 2015.

[27] A. Varga, "Omnet++," in *Modeling and Tools for Network Simulation*, K. Wehrle, M. Güneş, and J. Gross, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 35–59, isbn: 978-3-642-12331-3. doi: 10.1007/978-3-642-12331-3_3. [Online]. Available: https://doi.org/10.1007/978-3-642-12331-3_3.

[28] S. Kurkowski, T. Camp, and M. Colagrosso, "Manet simulation studies: The incredibles," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 9, no. 4, pp. 50–61, 2005, issn: 1559-1662. doi: http://doi.acm.org/10.1145/1096166.1096174. [Online]. Available: http://portal.acm.org/citation.cfm?id=1096166.1096174#.

[29] L. Bajaj, M. Takai, R. Ahuja, K. Tang, R. L. Bagrodia, and M. Gerla, "Glomosim: A scalable network simulation environment," 2002.

[30] G. Piro, L. A. Grieco, G. Boggia, F. Capozzi, and P. Camarda, "Simulating lte cellular systems: An open-source framework," *IEEE Transactions on Vehicular Technology*, vol. 60, no. 2, pp. 498–513, 2011. doi: 10.1109/TVT.2010.2091660.

[31] X. Chang, "Network simulations with opnet," in *Proceedings of the 31st Conference on Winter Simulation: Simulation—a Bridge to the Future - Volume 1*, ser. WSC '99, Phoenix, Arizona, USA: Association for Computing Machinery, 1999, pp. 307–314, isbn: 0780357809. doi: 10.1145/324138.324232. [Online]. Available: https://doi.org/10.1145/324138.324232.

[32] T. Givler and P. Lilienthal, "Using homer software, nrel's micropower optimization model, to explore the role of gen-sets in small solar power systems; case study: Sri lanka," National Renewable Energy Lab., Golden, CO (US), Tech. Rep., 2005.

[33] H. Hu, J. Zhang, X. Zheng, Y. Yang, and P. Wu, "Self-configuration and self-optimization for lte networks," *IEEE Communications Magazine*, vol. 48, no. 2, pp. 94–100, 2010. doi: 10.1109/MCOM.2010.5402670.

[34] G. Brockman, V. Cheung, L. Pettersson, *et al.*, *Openai gym*, 2016. arXiv: 1606.01540 [cs.LG].

[35] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html.

[36] V. Mnih, A. P. Badia, M. Mirza, *et al.*, *Asynchronous methods for deep reinforcement learning*, 2016. arXiv: 1602.01783 [cs.LG].

[37] T. P. Lillicrap, J. J. Hunt, A. Pritzel, *et al.*, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[38] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, *Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor*, 2018. arXiv: 1801.01290 [cs.LG].

[39] S. Fujimoto, H. van Hoof, and D. Meger, *Addressing function approximation error in actor-critic methods*, 2018. arXiv: 1802.09477 [cs.AI].

[40] M. Andrychowicz, F. Wolski, A. Ray, *et al.*, "Hindsight experience replay," in *Advances in neural information processing systems*, 2017, pp. 5048–5058.