FINAL YEAR PROJECT REPORT

MULTI AGENT REINFORCEMENT LEARNING FOR SUM-RATE MAXIMIZATION IN UAV ASSISTED IoT NETWORKS

By

A/C SYED ASAD ABBAS MEHDI

Pak/20095032, 95(B) EC

ADVISOR

SQUADRON LEADER DR SYED KHURRAM MAHMUD

CO-ADVISOR

WING COMMANDER ENGINEER ZAHID ALI

COLLEGE OF AERONAUTICAL ENGINEERING

PAF Academy, Asghar Khan, Risalpur

February 16, 2024

MULTI AGENT REINFORCEMENT LEARNING FOR SUM-RATE MAXIMIZATION IN UAV ASSISTED IoT NETWORKS

By

A/C SYED ASAD ABBAS MEHDI

Pak/20095032, 95(B) EC

ADVISOR

SQUADRON LEADER DR SYED KHURRAM MAHMUD

CO-ADVISOR

WING COMMANDER ENGINEER ZAHID ALI

Report submitted in partial fulfillment of the requirements for the degree of Bachelors of Engineering in Avionics, (BE Avionics)

In

COLLEGE OF AERONAUTICAL ENGINEERING

PAF Academy, Asghar Khan, Risalpur

February 16, 2024

Approval

It is certified that the contents and form of the project entitled “MULTI AGENT REINFORCEMENT LEARNING FOR SUM-RATE MAXIMIZATION IN UAV ASSISTED IoT NETWORKS” submitted by Aviation Cadet Syed Asad Abbas Mehdi have been found satisfactory for the requirement of the degree.

Advisor: SQUADRON LEADER DR SYED KHURRAM MAHMUD

Signature: ________________

Date: ________________

Co-Advisor: WING COMMANDER ENGINEER ZAHID ALI

Signature: ________________

Date: ________________

Dedication

I want to take this opportunity to express my sincere gratitude to all those who have played a significant role in making this project possible. I would like to extend a special thanks to my loving family and my dedicated advisor, whose unwavering support and guidance have been instrumental in my academic success. Without their encouragement and support, I would not have been able to reach this point. To my parents, in particular, I owe a debt of gratitude for their selfless sacrifices, endless encouragement, and unwavering belief in me. This report is a tribute to them and all those who have contributed to my journey, and I am honored to share this achievement with them.

Acknowledgement

I express my sincere gratitude to Allah Almighty, who bestowed upon me the strength and determination to complete this project to the best of my abilities. My parents, whose unfailing love, unrelenting support, and steadfast prayers have been the compass in my life, deserve the utmost gratitude. Without their support, this accomplishment would not have been possible. I am immensely grateful to my advisor, Squadron Leader Dr. Syed Khurram Mahmud, for his constant guidance, invaluable feedback, and unwavering support. His encouragement and mentorship have been instrumental in shaping my research skills and intellectual growth. I also extend my heartfelt thanks to my co-advisor, Wing Commander Engineer Zahid Ali, for his valuable input and for sharing his expertise in the field. I am grateful to all my teachers and colleagues who have contributed to my academic and professional growth in various ways. Finally, I would like to acknowledge the support of my friends and family members, who have been there for me throughout my academic journey. Thank you all for your support, encouragement, and motivation.

Abstract

Unmanned Aerial Vehicles (UAVs) have emerged as a promising solution to enhance the performance of Internet of Things (IoT) networks by providing dynamic and flexible communication support. Maximizing the sum rate, a critical metric representing the aggregate throughput of active devices, is essential for efficient resource allocation and improved network efficiency in UAV-assisted IoT systems. This paper explores the application of Multi-Agent Reinforcement Learning (MARL) to achieve sum rate maximization in UAV-assisted IoT networks. MARL enables UAVs to make coordinated decisions, and dynamically allocate resources based on the generated demand. The benefits of MARL in UAV-assisted IoT networks include optimizing energy-efficient data exchange, minimizing spectrum interference, and enhancing sum rate (total data rate) by dynamically adapting resource allocation and task distribution in UAV-assisted IoT networks. Despite challenges, such as UAV height, position, environment modeling, and training algorithm efficiency. MARL offers a promising avenue for optimizing sum rates and enabling innovative IoT applications in diverse and dynamic environments.

Contents

List of Figures **4**

List of Tables **5**

1 Introduction to the Project **6**

1.1 Project Title 6

1.2 Project Motivation 6

1.3 Project Description 6

1.4 Scope of the Project 7

1.5 Project Structure 8

1.6 Summary 9

2 Literature Review **11**

2.1 Basic Overview 11

2.2 Unfolding IoT: The evolution of connected devices 11

2.3 OMA-UAV Regime 12

2.4 Applications of OMA-UAV aided IoT networks 13

2.4.1 Surveillance and Reconnaissance 14

2.4.2 Communication and Connectivity Support 14

2.4.3 Agriculture and Environmental Monitoring 15

2.4.4 Target Tracking and Autonomous Swarm Operations 15

2.5 Key Performance Indicators(KPIs) 15

2.5.1 Sum-rate 15

2.5.2 Energy efficiency 16

2.5.3 Spectrum efficiency 16

2.6 Reinforcement Learning 16

2.6.1 Agent in Reinforcement Learning 18

2.6.2 Environment in Reinforcement Learning 19

2.6.3 State Mechanism in Reinforcement Learning 19

2.6.4 Action Mechanism in Reinforcement Learning 20

2.6.5 Reward Mechanism in Reinforcement Learning 20

2.6.6 Markov Decision Process in Reinforcement Learning 21

2.6.7 Q-Learning 22

2.6.8 Deep Q-Network (DQN) 23
2.6.9 Deep Deterministic Policy Gradient (DDPG) 23
2.7 Deep Learning 24
2.8 Neural Network 24
2.9 Optimizers 25
2.9.1 Adam 25
2.10 Activation Function 25
2.10.1 ReLU 25
2.10.2 Sigmoid 26
2.10.3 Softmax 26
2.11 Loss Function 26
2.11.1 Loss function for regression 26
2.11.2 Loss function for classification 27
2.12 Deep Q-Network Architecture 27
2.13 Chapter Summary 29

3 System Model 30

3.1 Methodology 30
3.2 Problem Formulation 31
3.2.1 Optimization Function 31
3.3 Environment Modelling 35
3.3.1 Introduction 35
3.3.2 Environment Initialization 36
3.3.3 Observation and Action Spaces 36
3.3.4 UAV Movement 36
3.3.5 Termination and Rewards 36
3.3.6 Data Rate Calculation 37
3.4 MARL Algorithm Framework 38
3.5 Neural Network Architecture (Q-Network Class) 39
3.6 DQN Agent Class 40
3.6.1 Neural Networks 40
3.6.2 Optimization 42
3.6.3 Hyperparameters 43
3.7 Training Procedure 43

3.8 Plotting and Analysis 43

4 Results And Discussion 45

4.1 Average Sum-Rate Comparison 45

4.2 Learning Rates: Navigating UAV Collisions 48

4.3 Energy Efficiency Comparison 51

4.4 Spectrum Efficiency Comparison 52

4.5 Deciphering DQN Learning Rates: A Loss Perspective 54

4.6 DQN Batch Size Analysis 57

5 Conclusion and Future Direction 59

A Program Code 62

A.1 Importing Libraries 62

A.2 Environment Modelling 62

A.3 DQN AGENT: Agent Network 66

A.4 DQN AGENT: Network Code 67

A.5 DQN AGENT: Training Procedure 69

A.6 Plotting Reward Curves 71

A.7 Plotting Loss Function (for Agent-0 for example) 72

A.8 Plotting Cumulative Sum Rate 75

A.9 Plotting Energy Eficiency 76

A.10 Plotting Spectrum Eficiency 77

A.11 Plotting Collision Rate 78

A.12 Analyzing Different Batch-Sizes With Best Learning Rate 0.001 78

B Bibliography 80

Bibliography 80

List of Figures

1 A flow chart representation illustrating the identification of challenges, proposed solutions, along with desirable and key performance indicators being provided. Additionally, frameworks for reinforcement learning are depicted. These visual aids are elaborated upon in later chapters, as detailed in section 1.5 of chapter 1. 10

2 UAV-assisted IoT application scenarios [11] 14

3 The relationship between AI, ML, RL, DL and DRL. Subdomains of machine learning include reinforcement learning. Deep learning facilitates both, supervised and reinforcement learning. 17

4 RL algorithms and classification 18

5 Illustration of RL paradigm with basic elements, such as agent, environment, reward, state and action [36]. 19

6 Illustration of a Markov Decision Process (MDP) diagram with five states (“1st S” to “5th S”) connected by actions (“a0” to “a4”). The “1st S” state has a reward labeled “rS” 21

7 Q-learning [7] 22

8 Deep Q-learning [7] 23

9 DDPG Actor-Critic Neural Networks [8] 24

10 ReLU Activation Function [37] 26

11 DQN Architecture 28

12 Methodology 30

13 Depiction of UAV assisted IoT network modelled for sum-rate maximization using Multi Agent Reinforcement Learning. 34

14 Environment Modelling of UAV assisted IoT netwroks. 35

15 Illustration of a multi-layer fully connected neural network architecture with 8 input features, 32 hidden neurons in the hidden layer with ReLU activation, and 5 output neurons. 39

16 Average reward comparison with its varying learning rates. 45

17 The figure shows the cumulative sum-rate (bps) achieved by our DQN agents when trained with three different learning rates of 0.1, 0.01, and 0.001 respectively. 47

18 The impact of different learning rates on UAV collisions. 48

19 The plot illustrates the impact of changing batch sizes on the sum rate achieved by DQN Agents. After optimizing the sum rate with the best learning rate (0.001), this analysis explores the performance sensitivity to different batch sizes. 50

20 Average energy efficiency comparison with its varying learning rates of DQN. 51

21 Average spectrum efficiency comparison with its varying learning rates of DQN. 53

22 Comparative Loss Analysis for DQN Agents with Learning Rates of 0.001, 0.01, and 0.1 54

23 Loss for DQN Agents with Learning Rate of 0.001 55
24 Loss for DQN Agents with Learning Rate of 0.01 55
25 Loss for DQN Agents with Learning Rate of 0.1 56
26 The plot illustrates the impact of changing batch sizes on the sum rate achieved by DQN Agents. After optimizing the sum rate with the best learning rate (0.001), this analysis explores the performance sensitivity to different batch sizes. 57

List of Tables

1 Environment Parameters 32

Chapter 1

1 Introduction to the Project

1.1 Project Title

The title of the project is ”Multi Agent Reinforcement Learning For Sum Rate Maximization In UAV Assisted IoT Networks”.

1.2 Project Motivation

RL-based methods are used because they can learn without needing specific models or data. The RL agent learns by trying out different actions, using rewards and penalties to figure out what works best [1]. However, these methods have limits on how much they can learn because they can only interact with the environment for a set amount of time. Despite this, they can still find good strategies for things like improving network performance, such as data transmission rates or energy and spectrum efficiency.

In UAV-assisted IoT networks, managing network resources is made more complicated by considering the UAV’s flight patterns and behavior [2]. This makes it harder to optimize everything together. Also, the network conditions can change, which makes it even trickier to find good strategies in a limited time.

To make RL methods work better in these situations, we can adjust how we manage large sets of data, change settings based on what’s happening, create better ways to measure success, design specific environments to learn in, and combine RL with other methods [24]. Doing these things can help improve how well RL works and, ultimately, how well the wireless network performs.

1.3 Project Description

In the era of wireless communication, the utilization of Unmanned Aerial Vehicles (UAVs) in Internet of Things (IoT) networks presents a promising avenue for improving communication efficiency. This project delves into the realm of multi-agent reinforcement learning to address the challenge of maximizing the sum rate in UAV-assisted IoT networks. The key objective is to develop intelligent agents that can autonomously learn optimal strategies to enhance the overall data transmission rates in dynamic and complex environments [1].

The project involves the design and implementation of a sophisticated multi-agent system, where each agent represents a UAV tasked with optimizing communication parameters to achieve the collective goal of maximizing the sum rate across the IoT network [2, 3]. Reinforcement learning algorithms will be employed to enable the agents to adapt and learn from their interactions with the environment, making informed decisions regarding communication protocols, routing strategies, and resource allocation.

1.4 Scope of the Project

The integration of unmanned aerial vehicles (UAVs) into wireless networks has brought about significant advancements due to their adaptability and widespread availability. With the rapid expansion of 5G services, the utilization of IoT devices has surged, necessitating efficient communication strategies. In this context, reinforcement learning (RL) emerges as a promising approach to enhance the sum-rate between UAVs and IoT users. By leveraging RL techniques, UAVs can dynamically adjust and optimize their data transmission rates, leading to improved performance and enhanced communication efficiency within UAV-enabled IoT networks.

This research focuses on optimizing the sum-rate within UAV-assisted IoT networks, where UAVs serve as relays between IoT network devices. Enhancing the sum-rate directly impacts both energy and spectrum efficiency within the network. To address this challenge, an RL-based approach is developed, aiming to find solutions within a limited number of learning cycles. The RL framework is specifically tailored to the operational scenario of UAVs, incorporating carefully designed Markov Decision Processes (MDPs), state and action mechanisms, and reward functions.

Additionally, the study examines the effects of adjusting RL hyper-parameters on network performance, providing insights into the interplay between reinforcement learning and network optimization.

1.5 Project Structure

This project report is organised as illustrated by the flow chart presented in Fig. 01 as followed:

Chapter 2 provides an extensive review of the literature, covering various aspects related to UAV-assisted IoT networks. It begins with a basic overview and then delves into the evolution and applications of IoT devices, particularly in the context of UAV support. The chapter also discusses key performance indicators (KPIs) such as sum-rate, energy efficiency, and spectrum efficiency. Moreover, it explores reinforcement learning (RL) techniques, including the agent, environment, state mechanism, action mechanism, reward mechanism, Markov Decision Process (MDP), Q-Learning, Deep Q-Network (DQN), and Deep Deterministic Policy Gradient (DDPG). Additionally, deep learning concepts such as neural networks, optimizers like Adam, activation functions like ReLU, Sigmoid, and Softmax, and loss functions for regression and classification are covered. The chapter concludes with a summary of the discussed topics, providing a comprehensive foundation for the subsequent chapters.

Chapter 3 presents the methodology employed in the research project, outlining the approach taken to optimize functions within the system model of UAV-assisted IoT networks. It details the process of modeling the environment, specifically focusing on the intricacies of UAV assistance in IoT networks. The chapter also introduces the Multi-Agent Reinforcement Learning (MARL) architecture framework designed for the project, which includes the implementation of the Deep Q-Network (DQN) algorithm. By synthesizing these components, the chapter provides a comprehensive framework for the optimization of system functions within UAV-assisted IoT networks. Finally, a summary encapsulates the key points discussed in the chapter, setting the stage for further exploration in subsequent sections.

Chapter 4 explores the results and discussions stemming from the analysis of network performance, specifically focusing on metrics such as sum-rate, energy efficiency, spectrum efficiency, collision rate, and cumulative sum rate. The chapter scrutinizes the impact of different learning rates of the Deep Q-Network (DQN) algorithm, varying between 0.001, 0.01, and 0.1, on the observed results. Through comparative analysis, the findings are contextualized, providing insights into the efficacy of the implemented methodologies and algorithms in optimizing the performance of UAV-assisted IoT networks.

Chapter 5 concludes this project report, which is followed by the future vision and applications.

1.6 Summary

The introduction of the project, titled ”Multi-Agent Reinforcement Learning for Sum Rate Maximization in UAV Assisted IoT Networks,” presents the motivation, description, scope, and structure of the project. The motivation behind the research lies in optimizing network performance within UAV-assisted IoT environments. The project aims to employ Multi-Agent Reinforcement Learning (MARL) to maximize the sum-rate, addressing challenges in energy and spectrum efficiency. The scope involves examining various applications and performance indicators, utilizing MARL architecture with Deep Q-Network (DQN) algorithms. The project structure delineates chapters covering literature review, methodology, results, and discussions. Through this comprehensive approach, the project seeks to contribute to enhancing communication efficiency in UAV-enabled IoT networks.

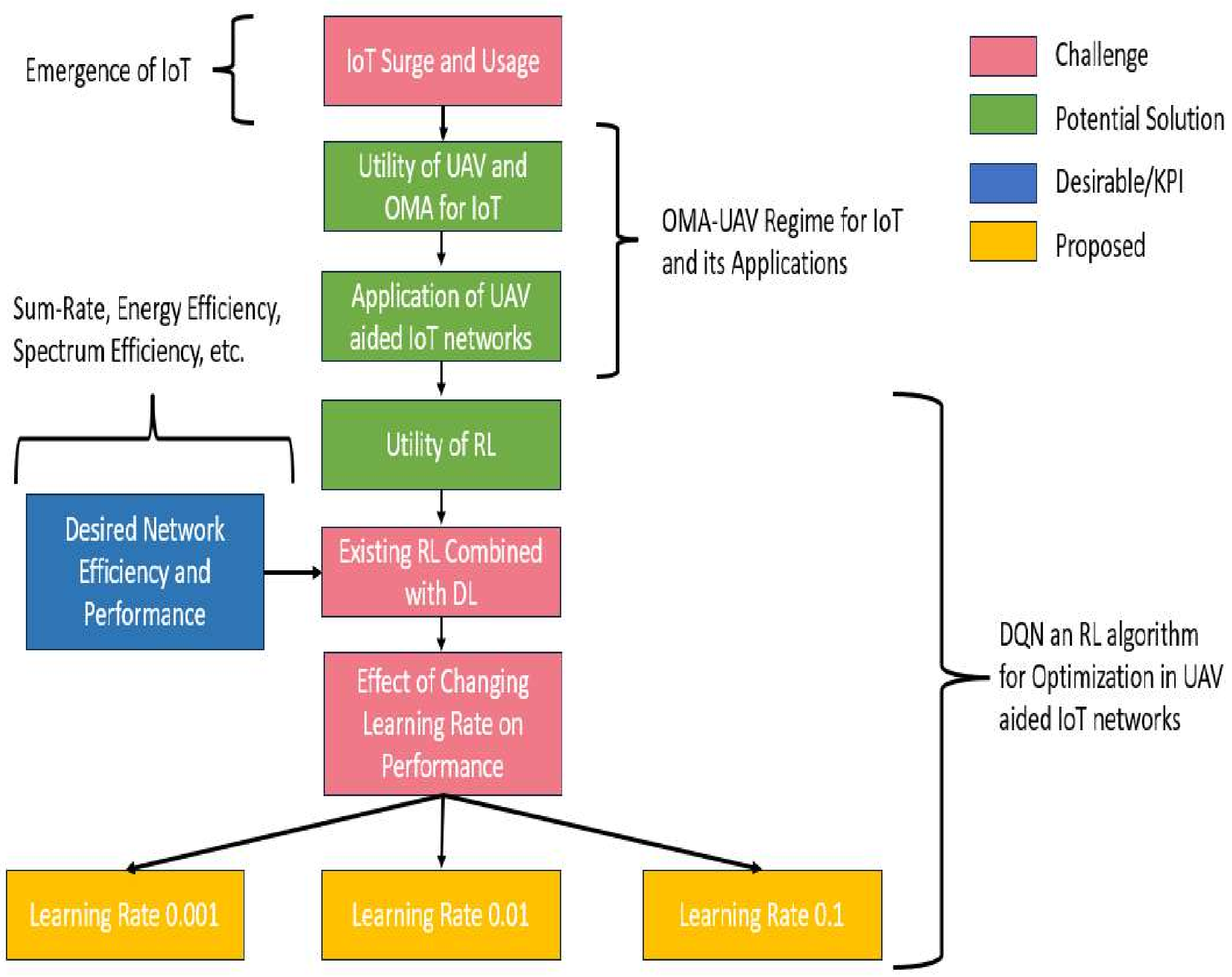
Emergence of IoT
IoT Surge and Usage
Utility of UAV and OMA for IoT
Application of UAV aided IoT networks
OMA-UAV Regime for IoT and its Applications
Sum-Rate, Energy Efficiency, Spectrum Efficiency, etc.
Utility of RL
Desired Network Efficiency and Performance
Existing RL Combined with DL
Effect of Changing Learning Rate on Performance
DQN an RL algorithm for Optimization in UAV aided IoT networks
Learning Rate 0.001
Learning Rate 0.01
Learning Rate 0.1
Challenge
Potential Solution
Desirable/KPI
Proposed

Figure 1: A flow chart representation illustrating the identification of challenges, proposed solutions, along with desirable and key performance indicators being provided. Additionally, frameworks for reinforcement learning are depicted. These visual aids are elaborated upon in later chapters, as detailed in section 1.5 of chapter 1.

Chapter 2

2 Literature Review

The chapter focuses on the utilization of unmanned aerial vehicles (UAVs) to assist in managing Internet of Things (IoT) networks using reinforcement learning (RL) frameworks for intelligent resource allocation. This chapter provides an overview of the technologies involved in the research and identifies potential challenges. It begins with a discussion on the emergence of IoT, followed by the use of UAVs and orthogonal multiple access (OMA) to support IoT networks, and its applications. The significance of sum-rate as a key performance indicator (KPI) in resource allocation scenarios is also addressed. Furthermore, the discussion covers the application of reinforcement learning in solving optimization problems involving OMA and UAVs, with a focus on existing deep RL frameworks and areas for potential performance improvement in UAV-assisted networks.

2.1 Basic Overview

Multi-agent reinforcement learning (MARL) can be applied to maximize the sum rate in UAV-assisted IoT networks. In this scenario, multiple UAVs act as agents, aiming to optimize the overall communication throughput in the network [12]. Using MARL techniques, the UAVs can learn to coordinate their actions, to maximize the sum rate while considering interference, channel conditions, and network constraints. MARL enables autonomous and adaptive decision-making processes in dynamic and complex UAV assisted IoT environments, leading to improved network performance.

2.2 Unfolding IoT: The evolution of connected devices

The 21st century is witnessing a profound transformation in how we engage with technology, primarily catalyzed by the emergence of IoT networks. These intricate webs of interconnected devices, sensors, and objects are revolutionizing communication and data exchange over the internet. Their integration into the fabric of fifth-generation (5G) and upcoming network infrastructures underscores their significance. The pivotal role of 5G in facilitating the widespread adoption of IoT cannot be overstated, as it delivers faster, more reliable, and lower-latency connectivity, along with increased bandwidth and capacity. This synergy between IoT and 5G is propelling advancements across diverse sectors, from industrial automation to healthcare, transportation, urban development, and beyond [1, 3].

As IoT permeates various industries, its impact on society continues to expand. Wearable devices are revolutionizing healthcare by enabling personalized monitoring and diagnostics, while autonomous vehicles are reshaping transportation systems, paving the way for safer and more efficient mobility. Moreover, the concept of smart cities, empowered by IoT technologies, promises to enhance urban living through real-time monitoring and optimization of infrastructure and services. With projections estimating the proliferation of IoT devices to surpass 25 billion by 2025, the momentum behind this technological paradigm shift is undeniable [3, 4, 5, 6, 7]. Looking ahead, further innovations in edge computing, artificial intelligence (AI), and next-generation connectivity, such as 6G, hold the promise of unlocking even greater possibilities within the ever-evolving IoT ecosystem, driving unprecedented growth and transformation.

2.3 OMA-UAV Regime

Orthogonal Multiple Access (OMA) is a crucial technology in the realm of wireless communication systems, particularly in the context of Unmanned Aerial Vehicle (UAV)-assisted Internet of Things (IoT) networks. OMA operates on the principle of allocating orthogonal resources to different users, allowing simultaneous transmissions without causing interference. This allocation ensures that each user's signal occupies a distinct portion of the spectrum, thereby enabling multiple users to access the network concurrently. The essence of OMA lies in its ability to mitigate interference among users, enhancing the overall efficiency and reliability of the communication system [19, 20, 21].

In the context of UAV-assisted IoT networks, OMA plays a pivotal role in optimizing resource allocation to meet the diverse requirements of IoT devices. The challenge lies in effectively managing the limited spectrum resources available for communication, while catering to the varying needs of different IoT applications [21, 23]. OMA addresses this challenge by allocating orthogonal resources, such as time slots, frequency bands, or codes, to IoT devices served by the UAV. By doing so, OMA ensures that each IoT device can transmit its data without causing interference to others, thereby maximizing the overall network capacity and throughput.

Mathematically, the sum rate generally in an OMA-UAV assisted IoT network can be expressed as:

$$R_{\text{sum}} = \sum_{i=1}^{N} \log_2 \left(1 + \frac{a_n.P_i|g_i|}{\sigma^2}\right)$$

Where:

- R_{sum} represents the sum rate of all IoT devices in the network.
- N denotes the total number of IoT devices.
- P_i represents the transmit power of the i^{th} IoT device.
- a_n represents the power coefficient.
- g_i denotes the channel gain between the UAV and the i^{th} IoT device.
- σ^2 represents the noise power.

2.4 Applications of OMA-UAV aided IoT networks

The UAV and IoT when combined together brings powerful technologies to enable a range of exciting applications and showcasing the potential of this innovative network paradigm. Figure 2 illustrates some of the main applications of UAV-assisted IoT networks, while some of them are described as followed.

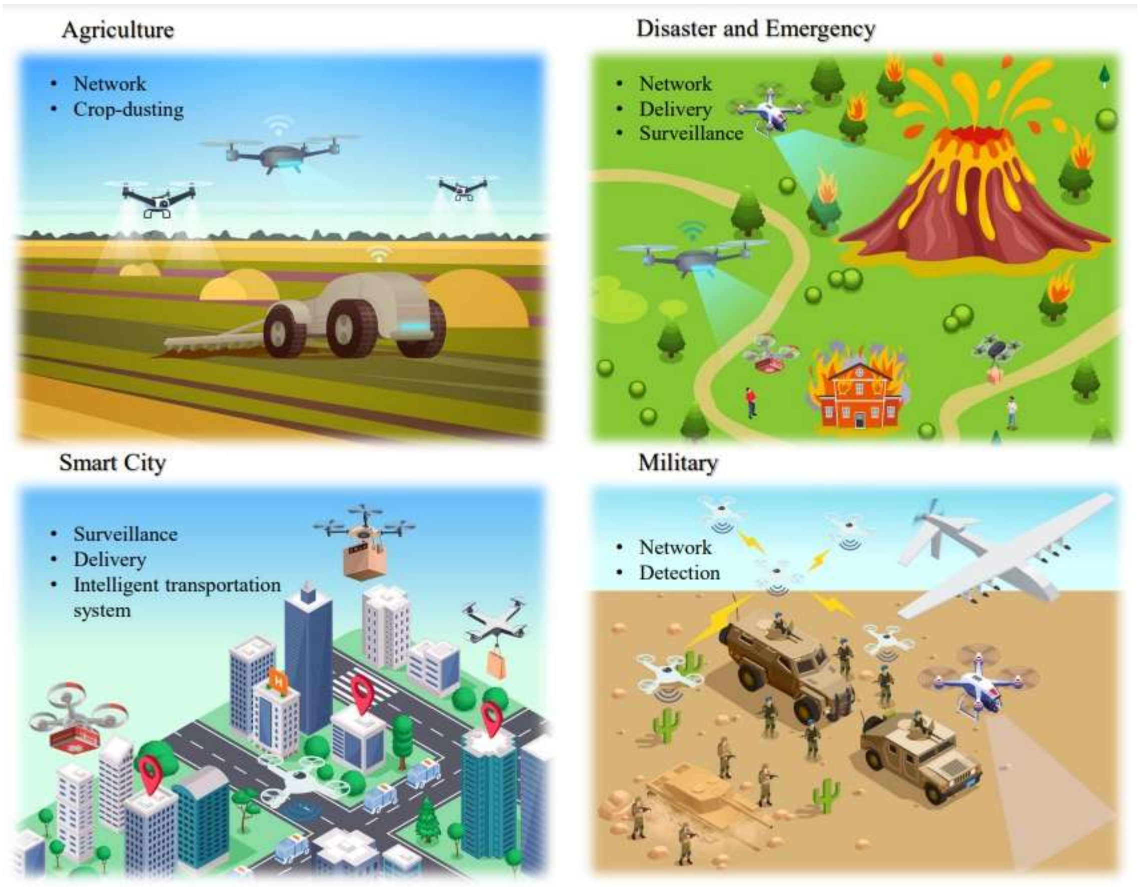

Figure 2: UAV-assisted IoT application scenarios [11]

2.4.1 Surveillance and Reconnaissance

These networks provide real-time intelligence, surveillance, and reconnaissance capabilities. Hence, facilitating battlefield monitoring, border security, disaster response, and environmental research [10].

2.4.2 Communication and Connectivity Support

This powerful network is ensuring continuous connectivity for military operations and supporting emergency responders and remote communities in areas with limited infrastructure [9].

2.4.3 Agriculture and Environmental Monitoring

UAV assisted IoT networks provide valuable assistance by using drones. The drones are equipped with sensors and cameras to collect data about crops, soil, and weather conditions. Fields are remotely monitored by farmers and decisions about irrigation, fertilization, and pest control, are made that lead to better crop yields. Hence, through this farmers can optimize their practices and enhance overall agricultural productivity and sustainability [10].

2.4.4 Target Tracking and Autonomous Swarm Operations

UAVs when equipped with can enable precise target tracking in military operations and can also be deployed for autonomous swarm operations. They also help serve civilian purposes like surveillance, traffic monitoring, and industrial asset management [9].

UAV assisted IoT networks demonstrate their versatility by offering seamless integration of technologies that is bringing significant advantages and enhancing various aspects of modern society and defense operations.

2.5 Key Performance Indicators(KPIs)

The project's key performance indicators (KPIs) are thoughtfully identified and carefully documented in the following section for comprehensive evaluation and efficient monitoring throughout the project's lifecycle.

2.5.1 Sum-rate

Sum-rate refers to the total data rate achieved by all agents in a network and indicates the collective capacity of the network to transmit data [24]. It is measured in bits per second (bps). The formula for the sum-rate is:

$$\text{Sum Rate} = \sum_{n=1}^{N} \frac{\omega}{K} \log_2(1 + \text{SINR}_n)$$

In above formula the SINR represents signal-to-interference-plus noise ratio. It measures the quality of a received signal, considering the influence of noise and interference power.

2.5.2 Energy efficiency

Energy efficiency means getting the most out of a system while using as little energy as possible, making it environmentally friendly and cost-effective.

$$EE = \frac{\text{Sum Rate}}{\text{Power of IoT}}$$

2.5.3 Spectrum efficiency

It refers to the ability of a communication system to utilize available wireless frequencies to enhance data transmission, and accommodate more users within the given spectrum resources.

$$EE = \frac{\text{Sum Rate}}{\text{UAV bandwidth}}$$

2.6 Reinforcement Learning

Reinforcement Learning (RL) has evolved into a widely adopted tool for optimizing systems and making decisions, often outperforming conventional optimization algorithms in dynamic and intricate settings [27, 28, 29]. It is a sub-domain of ML, which is a sub domain of artificial intelligence (AI) as shown in figure 3 [25, 26].

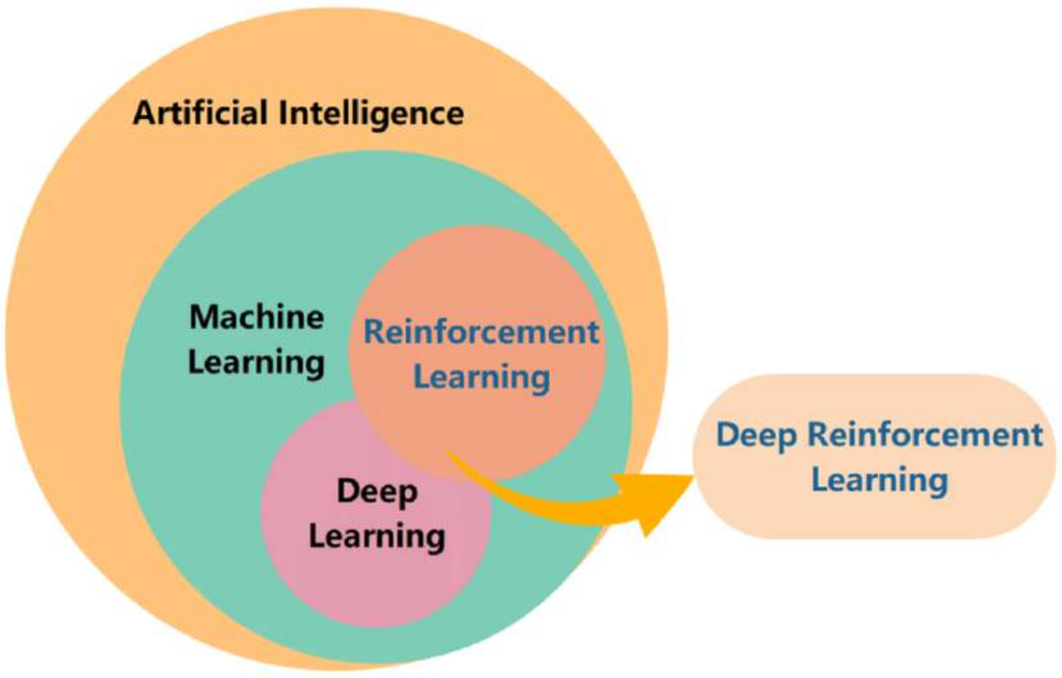

Figure 3: The relationship between AI, ML, RL, DL and DRL. Subdomains of machine learning include reinforcement learning. Deep learning facilitates both, supervised and reinforcement learning.

Its capacity to navigate complex environments makes it particularly relevant. RL has emerged as a pivotal technique in enhancing the efficacy of Unmanned Aerial Vehicles (UAVs) within the realm of IoT [2, 3]. Since its inception, RL has attracted substantial attention, wielding its robust data processing and analysis capabilities to imbue devices with intelligence, catalyzing transformative shifts across various industries [4, 5]. By seamlessly integrating RL into UAVs, their communication prowess, networking capabilities, and flight safety can be elevated, culminating in improved service quality within IoT applications [6].

Figure 4 illustrates the primary categories of RL algorithms and their classifications. RL can be categorized into two main branches: model-based RL and model-free RL. The former entails training models through data analysis to make predictions regarding unfamiliar data.

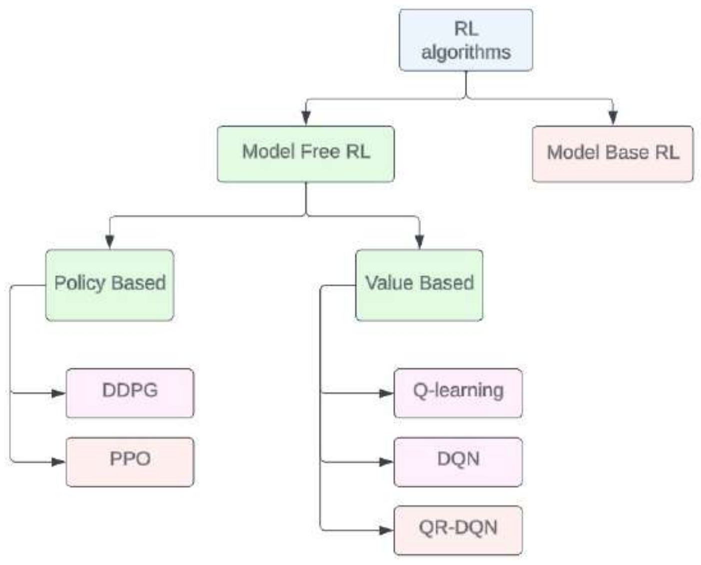

Figure 4: RL algorithms and classification

Moreover, this section entitles the theoretical aspects of reinforcement learning (RL). It will cover the fundamental elements of RL, including a discussion on temporal difference-based methods, Q-learning, deep Q-networks (DQNs), and Deep Deterministic Policy Gradient (DDPG).

2.6.1 Agent in Reinforcement Learning

In the reinforcement learning (RL) paradigm, the agent serves as the central learning entity. Its primary objective is to maximize rewards by selecting appropriate actions within a defined time horizon [24]. This process involves utilizing states through the action mechanism. For instance, in Figure 4, the agent earns a reward by executing action at time over state. Within the context of this project, the Unmanned Aerial Vehicle (UAV) is conceptualized as an agent operating within an Internet of Things (IoT) region. Its role is to make network-oriented decisions aimed at maximizing its reward. This instantaneous reward may be represented by metrics such as achieved communication rates which is sum rate. [24, 30]

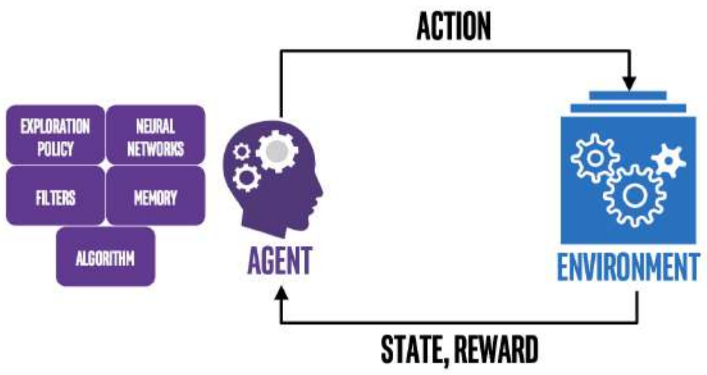

Figure 5: Illustration of RL paradigm with basic elements, such as agent, environment, reward, state and action [36].

2.6.2 Environment in Reinforcement Learning

In the context of reinforcement learning (RL), the environment encompasses a collection of physical or virtual elements within a physical world or simulated environment. These elements are engaged in interactions with the RL agent through a predefined set of actions. It's important to note that the actions performed by the agent do not have the capability to modify or influence the intrinsic dynamics of the environment. In scenarios where the environment is familiar, the outcomes of actions are anticipated beforehand [24, 30]. Conversely, in unknown environments, the outcomes of actions remain uncertain until executed. The RL agent evolves a policy by iteratively taking actions over states to grasp the underlying dynamics of the environment. For instance, consider an IoT region where a UAV acts as the agent, making decisions and interacting with its surroundings. The term ”environment” can also be interchangeable with the problem or challenge that the agent seeks to solve. Environments can exist in various forms: virtual (e.g., computer games or programs), physical (e.g., a UAV navigating an IoT region).

2.6.3 State Mechanism in Reinforcement Learning

In reinforcement learning, the state at a particular time step portrays the observations perceived by the agent. It can also signify the outcome of an action initiated by the agent. Each state or observation within the decision model of the environment is linked with either a stochastic or deterministic reward. These rewards stem from the underlying dynamics encapsulated within the environment model [24, 30]. For instance, when a UAV alters its position, it transitions to a different position, which is then regarded as a new state or observation.

2.6.4 Action Mechanism in Reinforcement Learning

In the setup of reinforcement learning, when the agent takes an action, it leads to a specific outcome. This outcome determines the state the agent ends up in and the reward it receives. The action mechanism is customized based on the characteristics of the environment or its model. For instance, in the earlier example, the action at a given time could involve the UAV adjusting its altitude while providing network services to ground equipment or users [24, 30].

2.6.5 Reward Mechanism in Reinforcement Learning

In reinforcement learning, the reward system plays a crucial role in shaping how the agent learns. When the agent takes an action in a certain state of the environment, it leads to a new state and a corresponding reward. This reward is based on specific performance metrics, with different parameters having different importance levels. The reward can encourage or discourage the agent from repeating a certain action in a given state [24, 30]. In the example mentioned earlier, the reward could be a network performance metric or a combination of multiple metrics, where some metrics carry more weight than others. The trend of rewards over time influences how the agent learns from its experiences. Designing the reward system is vital in determining the overall behavior and decisions of the agent as it interacts with the environment.

2.6.6 Markov Decision Process in Reinforcement Learning

Markov Decision Process (MDP) is like a game where agents have to make decisions to get the best outcome. It is like playing a game where agent move from one state to another, and at each state, it have different actions to take. However, the outcome of each action depends only on the current state agent is in, not on how it got there. So, agent can't change the past, and have to make decisions based only on what you know now [24, 30].

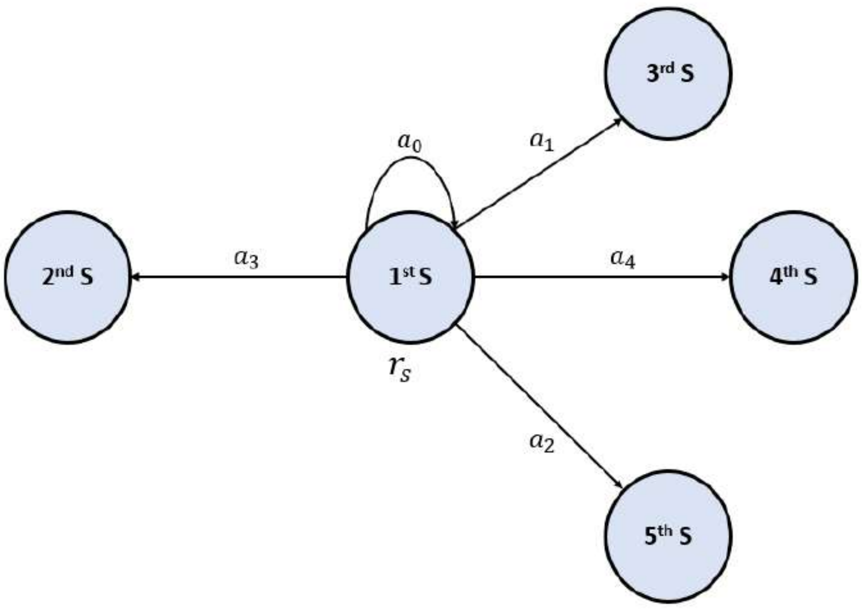

Figure 6: Illustration of a Markov Decision Process (MDP) diagram with five states ("1st S" to "5th S") connected by actions ("a0" to "a4"). The "1st S" state has a reward labeled "rS"

In an MDP, main goal is to find the best strategy or policy to maximize your rewards over time. Agent do this by learning which actions to take in each state to get the most reward in the long run. It's like figuring out the best moves to make in a game to win the most points. MDPs are used in many areas, like artificial intelligence, robotics, and economics, to help make decisions in situations where there's uncertainty and randomness involved. A Markov decision process comprises of following entities:

1. A set comprising all possible states S,
2. A set comprising all possible actions A,
3. Associated reward/utility function against each state as a consequence of an action taken is depicted as:
$$R(s_t, a_t) = E[r_{t+1}|s_t, a_t]$$
4. A discount factor γ.

2.6.7 Q-Learning

Q-learning is a reinforcement learning algorithm that learns the best actions to take in different situations. It does this by storing action-values in a table called the Q-table. The agent interacts with the environment, updating the Q-values based on rewards and transitions. Through exploration and exploitation, the agent learns to make decisions that maximize longterm rewards [24, 31]. During training, the agent uses the Q-learning update rule to adjust the Q-values, gradually converging towards the optimal policy. Once trained, the agent can use the learned Q-values to select actions that maximize expected rewards in each state, enabling it to make intelligent decisions to achieve a specific goal.

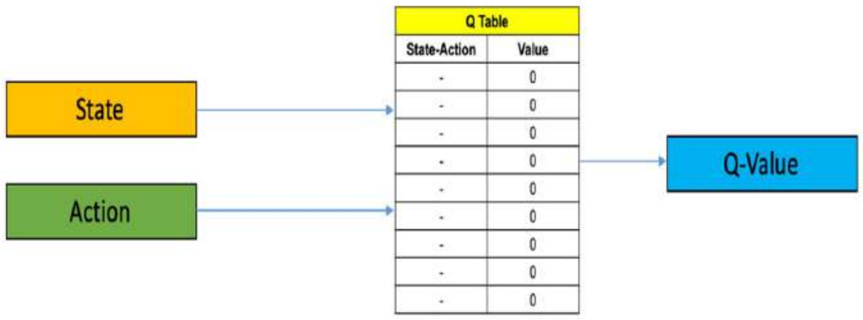

Figure 7: Q-learning [7]

2.6.8 Deep Q-Network (DQN)

Deep Q-Network is a reinforcement learning algorithm that combines Q-learning with deep neural networks. It employs a neural network, called the Q-network, to estimate the Q-values for different state-action pairs [24, 31]. DQN addresses the challenges posed by complex state spaces by leveraging deep learning techniques. The agent interacts with the environment, gathering experiences that are stored in a replay buffer. Periodically, the Q-network is trained using randomly selected experiences to update the Q-values. To ensure stability, a separate target network is used to compute the target Q-values.

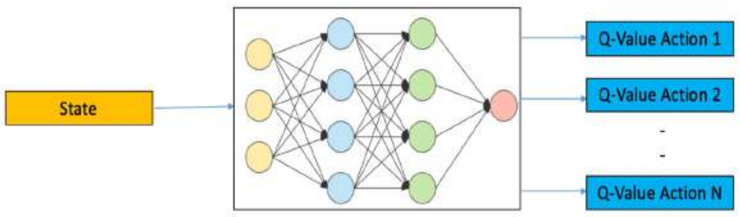

Figure 8: Deep Q-learning [7]

2.6.9 Deep Deterministic Policy Gradient (DDPG)

DDPG, which stands for Deep Deterministic Policy Gradient, is an actorcritic algorithm utilized in reinforcement learning. It combines the strengths of policy-based and value-based approaches [24]. The primary components of DDPG are deep neural networks, which are used to approximate both the policy (actor) and the Q-values (critic). The actor network is responsible for directly generating optimal actions given specific states, while the critic network evaluates the quality of these actions. Through iterative updates based on observed rewards, DDPG learns an optimal policy that aims to maximize cumulative rewards. This is achieved by adjusting the actor and critic networks to improve their performance. By employing deep neural networks, DDPG can handle complex and continuous action spaces, making it well-suited for tasks like robotic control and autonomous driving.[31]

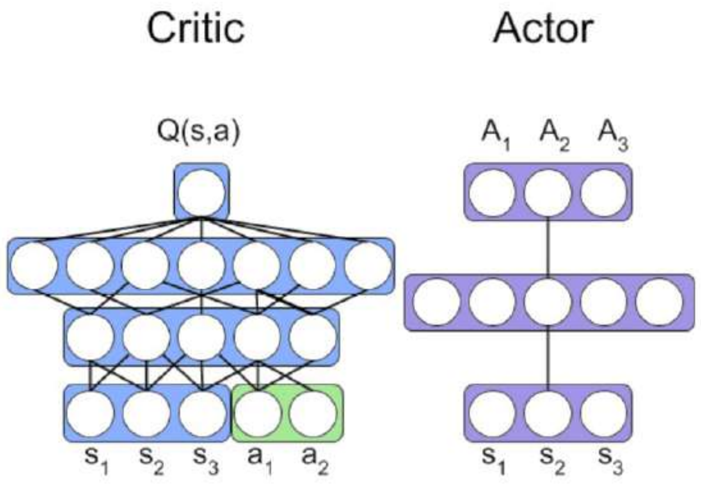

Figure 9: DDPG Actor-Critic Neural Networks [8]

2.7 Deep Learning

Deep learning is like a branch of machine learning. In this approach, we use deep neural networks to learn from labeled datasets. In reinforcement learning, we also use deep learning, and when we combine the two, we get something called deep reinforcement learning (DRL). In DRL, the deep neural network learns and understands different actions to take in different situations. We call this set of actions the policy [24, 30].

2.8 Neural Network

A neural network is a computational model inspired by the structure and functioning of the human brain. It's a type of machine learning algorithm that's particularly effective for tasks involving pattern recognition and classification. At its core, a neural network consists of interconnected nodes, or neurons, organized into layers. These layers include an input layer, one or more hidden layers, and an output layer. Each neuron in the network receives input signals from neurons in the previous layer, processes them, and produces an output signal that is passed on to neurons in the next layer.

2.9 Optimizers

An optimizer is crucial in training neural networks. It guides how the network learns and adjusts its weights and biases. The main aim is to minimize the loss function by updating parameters using gradients from backpropagation [37]. There are different optimizers, each with its own way of reducing the loss. Some adjust learning rates, others use momentum for faster convergence. Picking the right optimizer is vital as it affects training speed, convergence, and how well the model generalizes. One of the most common optimizer used in RL tasks include the Adam optimizer as explained below:

2.9.1 Adam

Adaptive Moment Estimation(Adam) optimizer uses adaptive learning rates to update the model parameters, which helps to achieve faster convergence and better generalization. Adam is a popular optimizer used in reinforcement learning tasks because of its easy implementation. Moreover, it is faster and calculates the exponential moving average of gradients [37].

2.10 Activation Function

The neuron represents the non-linear function applied to the input of the node, and that non-linear function is called the Activation function. Activation functions introduce non-linearity to the model, allowing it to learn more complex relationships and patterns. Common activation functions include ReLU (Rectified Linear Unit), Sigmoid, and Softmax.
A neural netwrok without the activation function is essentially just the linear regression.

2.10.1 ReLU

ReLU (Rectified Linear Unit) is the most commonly used activation function in neural networks. This function is simple and fast, thus making it a popular choice for deep neural networks.

ReLU is defined as R(x)= max(0, x):

$$R(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

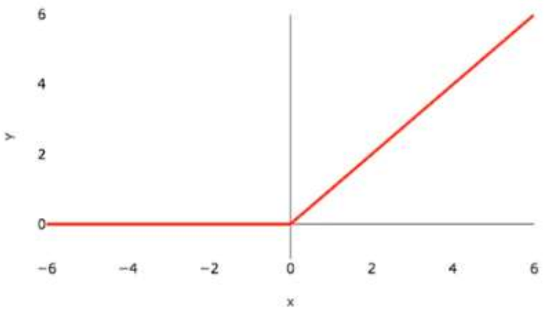

Figure 10: ReLU Activation Function [37]

2.10.2 Sigmoid

Sigmoid is an activation function usually used in the output layer of the binary classification, where result is either 0 or 1. As value for sigmoid function lies between 0 and 1 only so, results can be easily predicted to be 1 if value is greater than 0.5 and 0 otherwise.

2.10.3 Softmax

Softmax is applied to the last layer and only when we want the neural network to predict probability scores in the multi-class classification problems. Since, it produces values in the range of 0 to 1 , so they can represent probability scores.

2.11 Loss Function

Broadly, loss function can be classified into two major categories depending upon the type of learning task we are dealig with- Regression losses and Classification Losses.

2.11.1 Loss function for regression

- Mean Squared Error (MSE)

Mean Squared Error (also known as L2 loss is the average of the squared difference between the actual and predicted values. It is calculated using the formula:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

where n is the number of samples, y_i is the actual value, and $\hat{y}_i$ is the predicted value.

2.11.2 Loss function for classification

- Binary Cross Entropy Loss

This is the most common loss function that have two classes. It gives the probability value between 0 and 1 for a binary classification task. It calculates the average difference between predicted and actual classification.The binary cross-entropy formula is given by:

$$H(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)\right)$$

- Categorical Cross Entropy Loss

Categorical Cross Entropy Loss is essentially the Binary Cross Entropy Loss expanded to multiple classes. The categorical cross entropy loss formula is given by:

$$\text{Loss} = -\sum_{i}^{N} y_i \cdot \log(\hat{y}_i)$$

where N is the number of classes, y_i is the true probability distribution, and $\hat{y}_i$ is the predicted probability distribution.

2.12 Deep Q-Network Architecture

DQN, or Deep Q-Network, is a reinforcement learning algorithm that combines Q-learning with deep neural networks to enable learning directly from raw sensory input. It was introduced by Volodymyr Mnih and others in their seminal paper ”Playing Atari with Deep Reinforcement Learning" in 2013. [26]

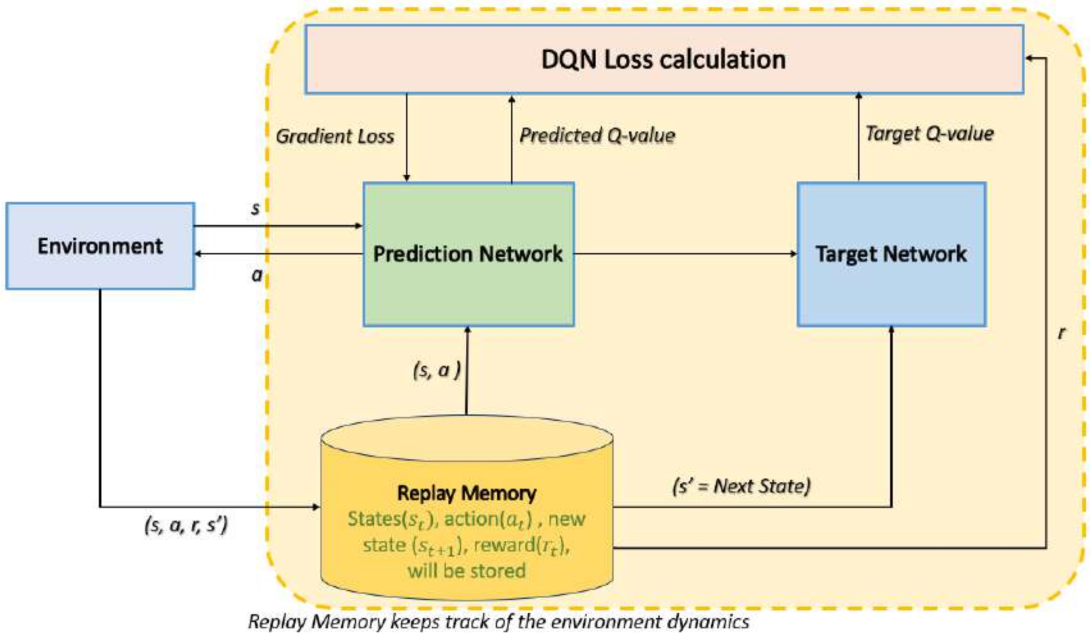

Figure 11: DQN Architecture

In our specific DQN architecture, the neural network consists of an input layer with eight neurons, representing an 1D array vector containing 8 features. This vector serves as the state representation for the agent. The network then features a hidden layer comprising 32 neurons, which learns to capture the complex relationships within the state space. The output layer, with five neurons corresponding to discrete actions (no movement, up, down, left, and right), produces Q-values for each action.

The network is trained using the Mean Squared Error loss function, and the training process involves experience replay, where batches of experiences are randomly sampled from a replay buffer, breaking temporal correlations between consecutive experiences. To enhance stability, a target Q-network, mirroring the main network, is employed, and its parameters are updated less frequently.

Additionally, the exploration-exploitation strategy follows an epsilon-greedy approach, where with probability epsilon, a random action is chosen, and otherwise, the action with the highest predicted Q-value is selected. This architecture enables our DQN to effectively learn and make decisions in an environment with 8-dimensional state input and 5 discrete action choices.

2.13 Chapter Summary

In this chapter, we traced the evolution of the Internet of Things (IoT) and its impact across industries, setting the stage for the discussion on Orthogonal Multiple Access (OMA) - Unmanned Aerial Vehicle (UAV) regimes. We emphasized OMA's significance in optimizing communication efficiency in IoT networks, particularly with UAV assistance. Exploring various applications of OMA-UAV setups, we highlighted their potential to enhance connectivity and coverage. Moving forward, we scrutinized key performance indicators (KPIs) crucial for evaluating the effectiveness of such networks, including sum rate, energy efficiency, and spectrum efficiency. Transitioning into reinforcement learning (RL), we delved into its core concepts and applications within dynamic environments like IoT networks. Furthermore, we explored the fusion of deep learning with RL to create Deep Reinforcement Learning (DRL), presenting neural network architectures, optimizers, activation functions, and loss functions as integral components. This comprehensive review sets the groundwork for further research and experimentation in the field, poised to unlock new advancements in IoT-enabled technologies.

Chapter 3

3 System Model

3.1 Methodology

The project is structured into distinct phases, as outlined below, ensuring a systematic approach towards its successful completion.

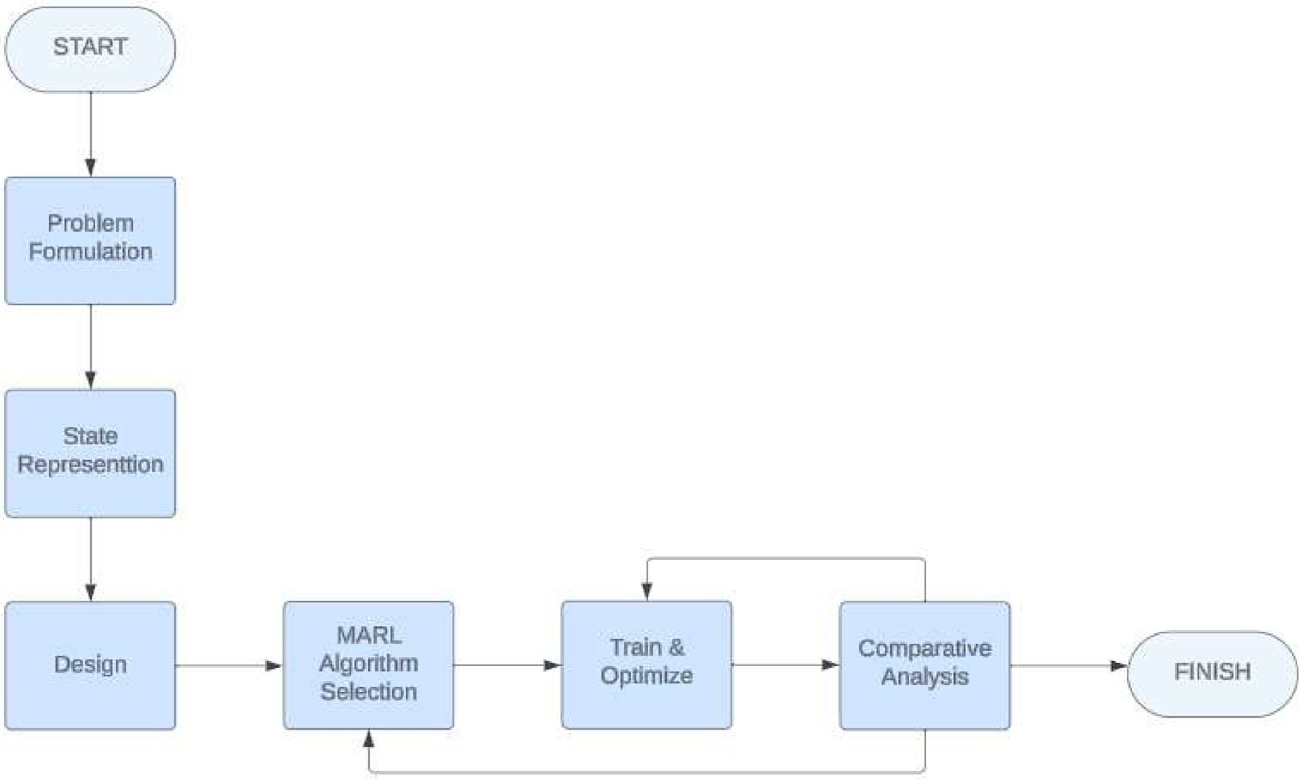

Figure 12: Methodology

Each phase will be carefully executed, allowing for efficient progress tracking and timely milestone achievements. The environment's characteristics are captured in a state representation, while a reward function is formulated to quantify improvements in sum rate based on UAV actions. A suitable MARL algorithm, such as DQN is chosen to train the agents. During training, UAVs interact with the environment, updating their policies through reinforcement learning techniques as mentioned. The trained agents are evaluated and fine-tuned, comparing their performance against baseline methods, and the MARL solution is eventually integrated. Hence, continuously improving sum rate performance in the UAV-assisted IoT ecosystem.

3.2 Problem Formulation

The contribution outlined in this section aims to enhance the sum-rate throughout the UAV's entire trajectory. This entails optimizing the sum-rate across both the UAV's flight trajectory and learning cycles while adhering to communication constraints associated with flight and Orthogonal Multiple Access (OMA). To address this intricate challenge, a model is formulated to capture the complexities of the problem, and a Reinforcement Learning (RL) framework is developed to optimize the sum-rate effectively.

3.2.1 Optimization Function

The objective function for maximisation of the net sum-rate is presented in this sub-section. The control variables and relevant network and flight constraints are depicted. The constraints pertain to minimum desired rate satisfaction.

This function aims to maximize the sum-rate, which is a measure of the total data transmission rate in the network. The objective function is defined below in equation 1:

$$\max_{\boldsymbol{\alpha_{1,1}},\boldsymbol{\alpha_{2,2}},\boldsymbol{x_u},\boldsymbol{y_u}} : \sum_{n=1}^{N}\sum_{u=1}^{U}\sum_{k=1}^{K}\beta_{k,u}SR_{k,u} \rightarrow \frac{\omega}{K}(\log_2(1+\text{SINR})) \tag{1}$$

The optimisation variables include:

C1:	$x_u \leq X$	Limitation for UAV to move within the x-axis
C2:	$y_u \leq Y$	Limitation for UAV to move within the y-axis
C3:	$h_u \leq H$	UAV height is fixed
C4:	$\omega_{k,u} \leq BW$	UAV Bandwidth can't be changed
C5:	$P_{\text{min}} \leq \alpha_{k,u} \leq P_{\text{max}}$	Power range for IoT devices

The network parameters, as detailed in Table 1, provide essential insights into the system setup. These parameters, including the UAV height, UAV bandwidth, grid size, noise power, power levels for IoT devices 1 and 2, and the path loss exponent for the free space model, play crucial roles in defining the system's characteristics and constraints, which directly influence the optimization variables previously elucidated.

Parameter	**Description**
Grid Size	The area under consideration is a square grid with dimensions of $10\,\text{m} \times 10\,\text{m}$.
UAV Height	The altitude at which the unmanned aerial vehicle (UAV) operates is $10\,\text{m}$.
Bandwidth (ω)	The available bandwidth for communication is $10\,\text{kHz}$.
Noise Power	The level of noise power in the system is $-90\,\text{dBm}$.
Path Loss Exponent (β)	The path loss exponent, representing the rate of signal decay with distance, is 2.3.
P_{min}	The minimum transmission power level is $-20\,\text{dBm}$.
P_{max}	The maximum transmission power level is $30\,\text{dBm}$.

Table 1: Environment Parameters

The optimization problem involves finding optimal values for the following control variables:

- $\boldsymbol{\alpha_{1,1}}$, $\boldsymbol{\alpha_{2,2}}$: Power allocation for IoT devices 1 and 2, respectively.
- $\boldsymbol{x_u}$, $\boldsymbol{y_u}$: These represent the coordinates of the UAV's position in the grid.

The objective is to maximize the following expression:

$$\sum_{n=1}^{N}\sum_{u=1}^{U}\sum_{k=1}^{K}\beta_{k,u}\cdot SR_{k,u} \rightarrow \frac{\omega}{K}\cdot(\log_2(1+\text{SINR})) \tag{2}$$

where:

$$\sum_{n=1}^{N}\sum_{u=1}^{U}\sum_{k=1}^{K} \rightarrow \ldots$$

- N is the number of episodes.
- U is the number of UAVs.
- K is the number of IoT devices.

Moreover, the binary matrix representation delineates the connectivity between UAVs and IoT devices. Each element in the matrix reflects whether a UAV is linked to a specific IoT device. For instance, the identity matrix:

$$\begin{bmatrix}\beta_{1,1} & \beta_{2,1}\\ \beta_{1,2} & \beta_{2,2}\end{bmatrix} \rightarrow \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} \tag{3}$$

It indicates that only the first UAV is connected to the first device, and the second UAV is linked to the second device as shown in Figure 12. Conversely, the absence of a connection is denoted by zero.

The constraints imposed by this binary representation ensure the integrity of the communication network. The summation of all elements in the matrix signifies the total number of active connections, ensuring it does not exceed 2, denoted by:

$$\sum_k \sum_u \beta_{k,u} \leq 2 \tag{4}$$

This further explains that the summation across UAVs for each IoT device guarantees that at most one UAV is connected to any given device:

$$\sum_u \beta_{k,u} \leq 1 \tag{5}$$

This framework optimizes the utilization of resources, facilitating efficient uplink transmission while maintaining connectivity constraints. This optimization function aims at maximizing sum-rate while considering various constraints related to movement along both axes, fixed height, unchangeable bandwidth and power range limitations.

Figure 13: Depiction of UAV assisted IoT network modelled for sum-rate maximization using Multi Agent Reinforcement Learning.

Each term and constraint plays a pivotal role in ensuring that the optimization does not only focus on maximizing sum-rate but also adheres to physical and technical limitations inherent in such networks. This balance between maximization and constraint adherence is key to the successful application of this function in real-world scenarios.

3.3 Environment Modelling

3.3.1 Introduction

A custom environment was build for simulating the movement and communication of Unmanned Aerial Vehicles (UAVs) in a grid-based scenario. This environment is designed to model interactions between UAVs and ground devices, considering factors such as UAV movement, device locations, device powers, and communication bandwidths. The primary goal is to calculate the data rate achieved by each UAV based on its position relative to ground devices and the given communication parameters.

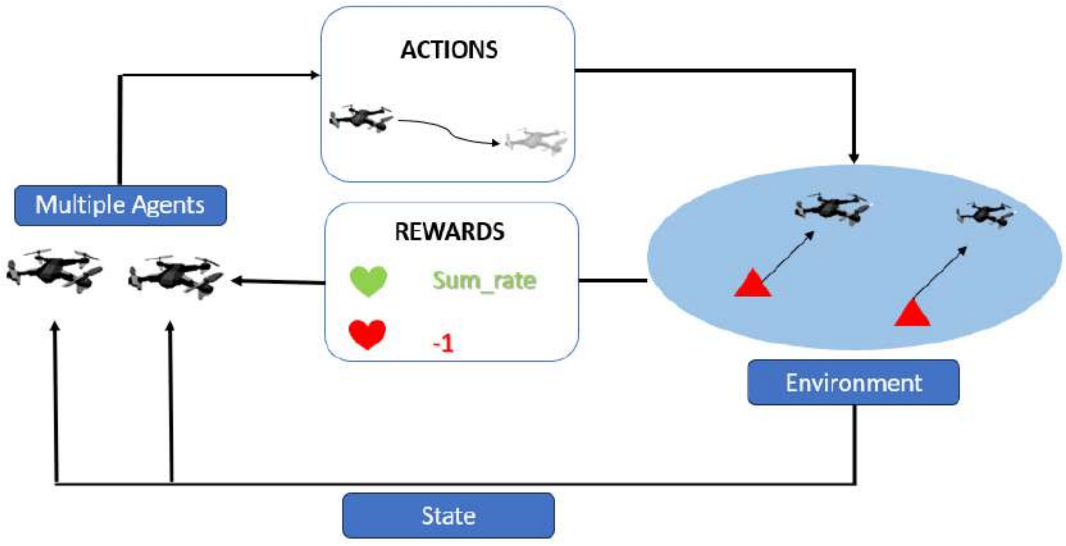

Figure 14: Environment Modelling of UAV assisted IoT netwroks.

3.3.2 Environment Initialization

The environment is initialized with a grid of a specified size (default is 10x10), where UAVs and ground devices are randomly positioned. Each UAV is associated with a unique identifier (agent-0 and agent-1 in this case). The number of UAVs, ground devices, and their respective initial configurations are set within the environment.

3.3.3 Observation and Action Spaces

The observation space is defined as a Box space with a low value of 0, a high value of 500, and a shape of (8). This space encompasses information about UAV bandwidths, the positions and bandwidths of other UAVs, and the powers of ground devices. The action space is Discrete, representing five possible actions: no movement and movement in four cardinal directions (up, down, left, right).

3.3.4 UAV Movement

During each step, UAVs can take actions to move within the grid. The movement is determined by the actions taken, where 0 represents no movement, and 1 to 4 correspond to moving up, down, left, and right, respectively. UAV positions are updated accordingly, with boundary checks to ensure they stay within the grid.

3.3.5 Termination and Rewards

The environment has a termination condition based on the number of steps taken, with a default limit of 100 steps. If this limit is reached, the episode is terminated for all agents. Collisions between UAVs are detected, and if a collision occurs, a negative reward of -1 is given to the respective UAV. The collision information is recorded in the environment's info dictionary.

3.3.6 Data Rate Calculation

The data rate for each UAV is calculated based on its position relative to ground devices. The key factors influencing the data rate include the distance between the UAV and the nearest ground device, the device's power, and the communication bandwidth of the UAV. The Signal-interference-to-Noise Ratio (SINR) is calculated using the formula:

$$\text{SINR} = \frac{\alpha_n \cdot \text{Device Power} \cdot \text{Channel-gain}}{\text{Thermal Noise}}$$

where:

- α_n is Power Coefficient,
- Device Power is the power of the ground device,
- Noise is a predefined noise value.

The distance between UAV and device, denoted as channel gain. It is calculated using the ground distance and UAV height. The path loss is modeled using a power-distance relation.

$$\text{Distance} = \frac{1}{\sqrt{(\text{UAV}_x - \text{Device}_x)^2 + (\text{UAV}_y - \text{Device}_y)^2 + \text{UAV Height}^2}^{\,2.3}}$$

where:

- UAV_x and UAV_y are the x and y coordinates of the UAV, respectively,
- Device_x and Device_y are the x and y coordinates of the ground device, respectively,
- UAV Height is the height of the UAV,
- 2.3 is the value of path loss exponent.

The data rate is then determined using the Shannon-Hartley theorem:

$$\text{Data Rate} = \frac{\text{UAV Bandwidth}}{N} \cdot \log_2(1 + \text{SINR})$$

3.4 MARL Algorithm Framework

Choosing an appropriate MARL algorithm or combination of algorithms to enable decision-making among the UAV agents. Common approaches include Q-learning, Deep Q-Networks (DQN), or actor-critic methods (DDPG). Consider the scalability and computational requirements of the environment DQN was selected.

The Deep Q-Network (DQN) algorithm is a reinforcement learning technique used for training agents to make decisions in environments with discrete action spaces. Below is a comprehensive explanation of the DQN algorithm framework, used in the optimizing the sum rate.

- Utilizes experience replay, where past experiences are stored in a replay buffer and sampled randomly during training. This helps break the temporal correlation between consecutive samples and reduces the likelihood of the model getting stuck in local minima [32].

- Employs a target network, which is a separate copy of the Q-network used for computing target Q-values during training [24, 31]. This helps stabilize the training process by reducing the oscillations in Q-values.

- Balances exploration and exploitation by using an epsilon-greedy strategy, where the agent chooses a random action with probability epsilon. This allows the agent to explore the state space effectively [31, 32].

3.5 Neural Network Architecture (Q-Network Class)

The neural network used in the DQN algorithm is implemented through the Q-Network class. This feed-forward neural network consists of three layers: an input layer, a hidden layer, and an output layer as illustrated in Fig. 15. The input layer has neurons equal to the dimensionality of the state space, which is a 1D vector array with 8 features, then the number of neurons in the input layer would be 8. Each neuron in the input layer corresponds to one feature in the state vector, creating a direct mapping between the input neurons and the individual components of the state.

The hidden layer contains 32 neurons with Rectified Linear Unit (ReLU) activation, and the output layer has neurons corresponding to the number of possible actions, which is 5. This architecture enables the network to approximate the Q-function, predicting Q-values for each action given a particular state.

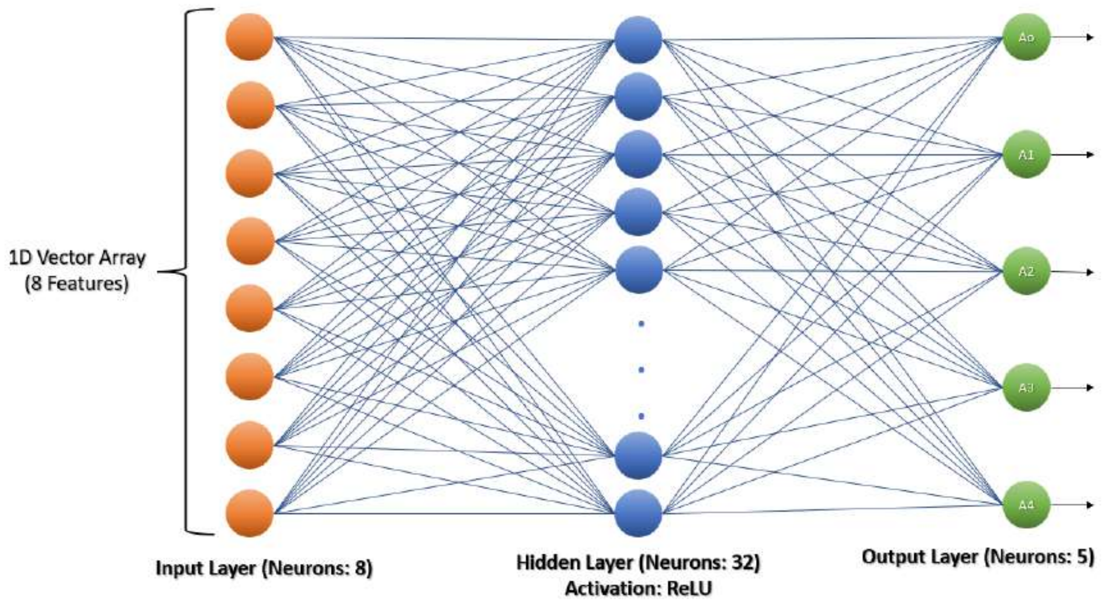

Figure 15: Illustration of a multi-layer fully connected neural network architecture with 8 input features, 32 hidden neurons in the hidden layer with ReLU activation, and 5 output neurons.

3.6 DQN Agent Class

The DQN Agent class initializes the DQN agent with essential components:

3.6.1 Neural Networks

The agent maintains two instances of the Q-Network – prediction model and target model. The prediction model is the local Q-network that gets updated during training, while the target model is a target Q-network used for stability in learning. As explained below are the equations used in building the DQN agent network.

1. **Soft Update of Target Model Parameters**:

$$\theta_{\text{target}} = \tau \cdot \theta_{\text{local}} + (1 - \tau) \cdot \theta_{\text{target}} \tag{6}$$

   - **Purpose**: This equation updates the parameters of the target neural network slowly towards the parameters of the local (online) neural network to increase stability.

   - **Variables**:
     - θ_{target}: Parameters of the target model.
     - θ_{local}: Parameters of the local model.
     - τ: Soft update parameter ($0 < \tau < 1$).

2. **Epsilon-Greedy Policy for Action Selection**:

$$\text{action} = \begin{cases} \text{random action} & \text{with probability } \epsilon \\ \text{argmax}(Q(s)) & \text{with probability } 1 - \epsilon \end{cases} \tag{7}$$

- **Purpose**: This equation chooses actions according to an epsilon-greedy strategy, balancing exploration and exploitation.

- **Variables**:
  - ϵ: Exploration-exploitation trade-off parameter.
  - $Q(s)$: Q-values for the given state s.

3. **Q-Learning Target**:

$$Y = R + \gamma \cdot (1 - \text{done}) \cdot \max_{a'}(Q_{\text{target}}(S', a')) \tag{8}$$

- **Purpose**: This equation computes the target Q-value used for training the neural network.

- **Variables**:
  - Y: Target Q-value.
  - R: Immediate reward.
  - γ: Discount factor.
  - done: Boolean indicating whether the episode has terminated.
  - $Q_{\text{target}}(S', a')$: Q-values of the target model for the next state S' and all possible actions a'.

4. **Loss Calculation using Mean Squared Error (MSE)**:

$$\text{loss} = \text{MSE}(Q(s, a), Y) \tag{9}$$

- **Purpose**: This equation computes the loss function, which quantifies the difference between predicted Q-values and target Q-values.

- **Variables**:
  - loss: Loss value.
  - $Q(s, a)$: Predicted Q-value for state s and action a.
  - Y: Target Q-value.

5. **Update of Epsilon for Exploration-Exploitation Trade-off**:

$$\epsilon = \max(\epsilon_{\text{min}}, \epsilon - \epsilon_{\text{decay}}) \tag{10}$$

- **Purpose**: This equation updates the exploration-exploitation trade-off parameter (ϵ) over time.

- **Variables**:
  - ϵ: Exploration-exploitation trade-off parameter.
  - ϵ_{min}: Minimum value for ϵ.
  - ϵ_{decay}: Decay rate for ϵ.

These equations are fundamental components of the DQN algorithm, responsible for action selection, target computation, loss calculation, and exploration strategy.

3.6.2 Optimization

The **Adam** optimizer is utilized for optimization, employing distinct learning rates (lr) of 0.1, 0.01, and 0.001. This choice of learning rates influences the agent's ability to adapt and learn from experiences during the training process. Different learning rates can impact the convergence speed and overall performance of the neural network.

3.6.3 Hyperparameters

Hyperparameters Used in DQN code are as follows:

- Learning Rate (lr): **0.1, 0.01, 0.001**
- Discount Factor (gamma): **0.99**
- Exploration-Exploitation Trade-off (epsilon): **1.0**
- Minimum Exploration Probability (epsilon_min): **0.01**
- Batch Size: **64**
- Replay Buffer Size: **1000**

Each hyperparameter plays an important role in shaping the behavior and performance of the DQN agents during training.

3.7 Training Procedure

The training procedure involves creating an environment (assumed to be a custom UAV environment in this context) and initializing two DQN Agent instances with different learning rates. The agents are then trained over a specified number of episodes, and the best models for each agent are saved based on the average sum-rate achieved during training.[24]

3.8 Plotting and Analysis

The code has been provided with the features to create plots showing how well the agents are learning. These visualizations help us understand how the rewards, energy, and spectrum efficiencies and losses change during the training episodes. Moreover, to provide a clear picture of how effective the learning rates are for the agents the analysis is performed in the next chapter.

In a nutshell, the DQN algorithm relies on a neural network to estimate the Q-function, ensuring more stable training through experience replay, and enhances convergence with a target network. The training strategy involves finding a balance between exploring new actions and exploiting known ones to efficiently learn the best strategies in a given environment.

Chapter 4

4 Results And Discussion

In this section we present the results on the performance of proposed framework analysis. The comparison involves evaluating the sum-rate achieved by the DQN algorithm under three distinct learning rates: 0.001, 0.01, and 0.1. Subsequently, using these same learning rates, we compute the energy efficiency and spectrum efficiency. This comparative analysis aims to provide insights into the impact of different learning rates on the overall performance of our UAV assisted IoT environments.

4.1 Average Sum-Rate Comparison

The average sum rate comparison over the course of OMA is depicted in Fig. 16.

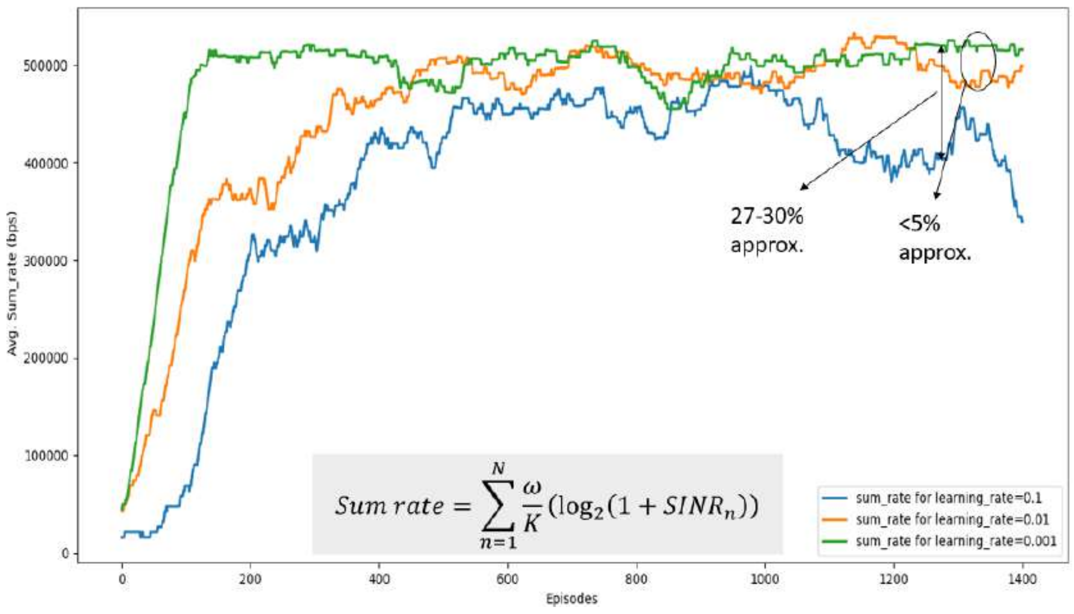

Figure 16: Average reward comparison with its varying learning rates.

The graph compares the sum rate achieved by DQN agents trained with three different learning rates: 0.1, 0.01, and 0.001. The sum rate is plotted against the number of episodes, which likely represents the amount of training the agents have undergone.

Here's a comparative analysis of the three learning rates based on the graph:

1. **Learning Rate 0.1 (Blue Line):** This learning rate does not shows a rapid initial increase in the sum rate, and lies at a performance gap of **27-30% approx** with maximum reward gained learning rate. However, it stabilizes at a lower sum rate compared to the other two learning rates. The blue bar represents a learning rate of 0.1 and shows a overall sum-rate reaches approximately 570,000 bps. This suggests that a higher learning rate might lead to smaller convergence initially and will not result in the best long-term performance.

2. **Learning Rate 0.01 (Orange Line):** This learning rate shows a slight performance gap of greater than **5%** approximately and a moderate growth.To a learning rate of 0.01, shows a cumulative sum rate of around 575,000 bps. Moreover, it eventually surpasses the sum rate of the learning rate 0.1, indicating better performance with more episodes. This suggests that a moderate learning rate might be a good balance between speed of learning and long-term performance.

3. **Learning Rate 0.001 (Green Line):** This learning rate increases slowly but consistently achieves the highest sum rate and gets close to 580,000 bps approximately as shown in the Fig. 17. This suggests that a smaller learning rate is faster to learn initially, and leads to better performance and convergence over more episodes. Hence, setting the bench mark for this analysis.

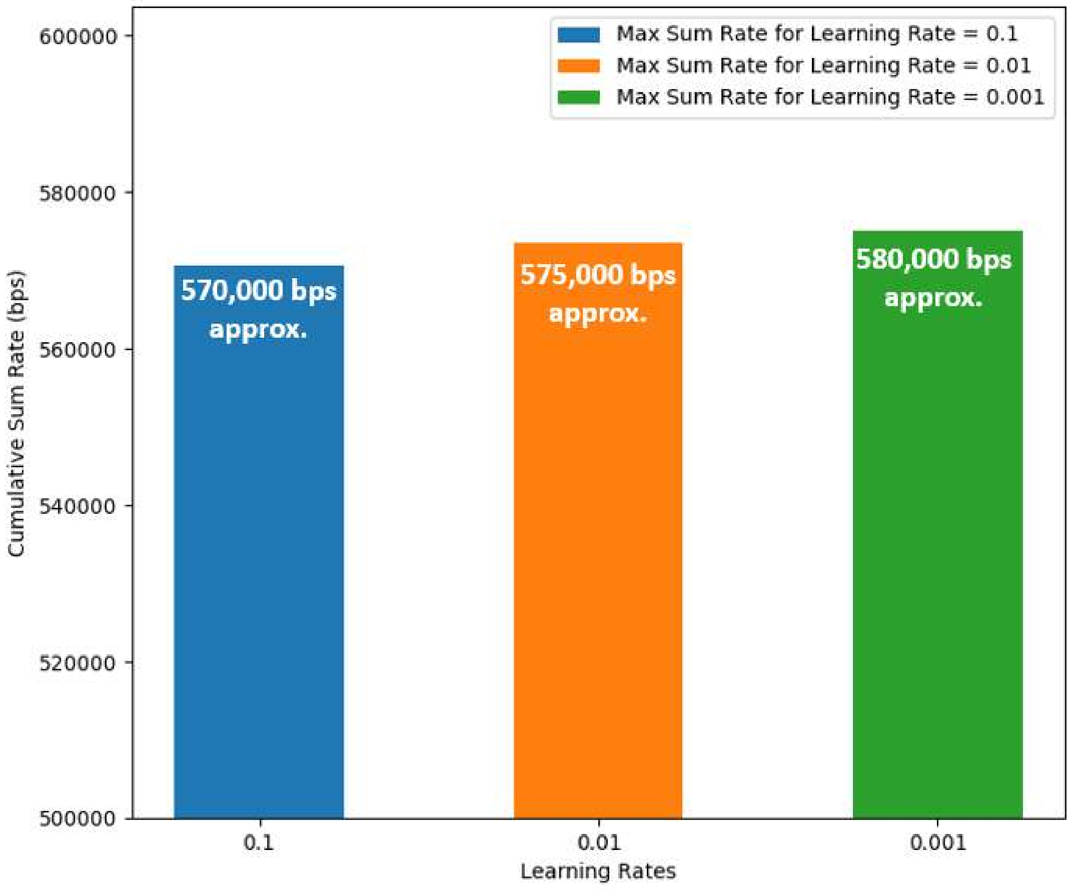

Figure 17: The figure shows the cumulative sum-rate (bps) achieved by our DQN agents when trained with three different learning rates of 0.1, 0.01, and 0.001 respectively.

In conclusion, the selection of the learning rate is paramount for achieving optimal performance in our UAV-assisted IoT network environment. It is observed that a smaller learning rate results in better performance metrics, including a higher maximum sum rate, indicating more efficient utilization of the spectrum. However, these observations are specific to the conditions of this particular experiment, and the optimal learning rate may vary under different scenarios or parameters. Therefore, a careful balance between convergence speed and performance stability is crucial. This approach ensures robust and efficient training of DQN agents, ultimately leading to improved performance in UAV-assisted IoT networks.

4.2 Learning Rates: Navigating UAV Collisions

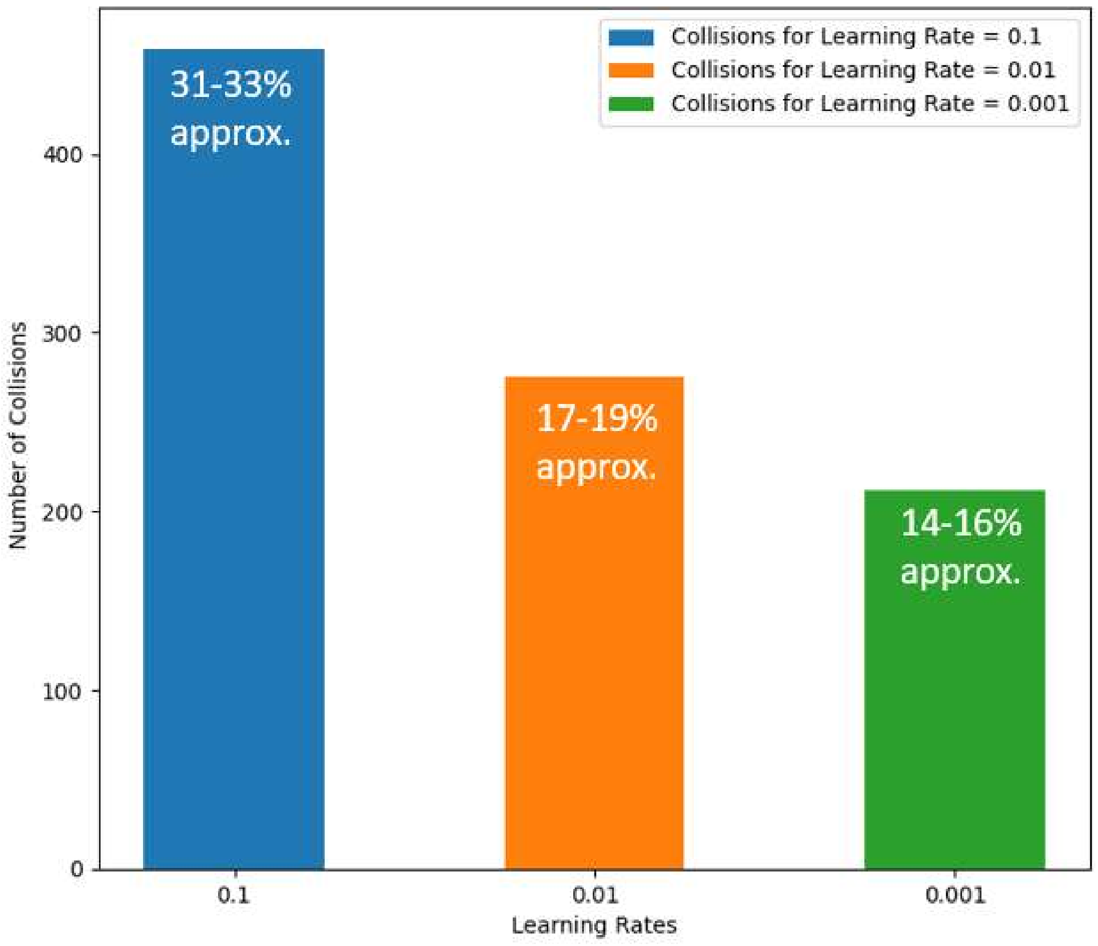

Figure 18: The impact of different learning rates on UAV collisions.

The bar plots the number of collisions made by Unmanned Aerial Vehicles (UAVs) when trained with three different Deep Q-Network (DQN) learning rates in UAV assisted IoT networks. The blue bar, representing a learning rate of 0.1, shows the highest number of collisions, reaching close to 500 **(31-33% approx.)**. This indicates that a higher learning rate leads to more collisions, suggesting that the UAVs are not learning effectively.

In contrast, the orange and green bars represent lower learning rates of 0.01 and 0.001 respectively. The orange bar is significantly lower than the blue one but still considerable in height as it get a total of **17-19%** collision rate approximately, indicating a reduction in collisions but not optimal performance.

The green bar is the shortest among them all, showing that at a learning rate of 0.001, both UAVs have minimal collision rate of **(14-16% aprrox.)** which implies enhanced performance and safety in navigation. This comparative analysis suggests that a lower learning rate results in fewer collisions, thus leading to safer and more efficient navigation for UAVs in IoT networks.

$$\text{No. of Collisions} = \sum_{c=0}^{c=1499} \left(\text{Collisions} \cdot I\left[(\text{Rate}_{\text{IoT1}} < \text{Rate}_{\text{QoS}}) \vee (\text{Rate}_{\text{IoT2}} < \text{Rate}_{\text{QoS}})\right]\right)$$

The equation represents the calculation of the total number of collisions occurring in a system over a specified period, denoted as ”No. of Collisions.” This total is determined by summing the individual collision events, indexed by c from episodes 0 to 1499. Each collision event is evaluated using an indicator function, denoted by I, which returns 1 if the condition inside the brackets is true and 0 otherwise. The condition checks whether the rate of data transmission for either IoT device 1 or IoT device 2, represented by $\text{Rate}_{\text{IoT1}}$ and $\text{Rate}_{\text{IoT2}}$ respectively, is less than the quality of service threshold (Rate_{QoS}). If either device’s transmission rate falls below the threshold, it indicates a collision event, contributing to the total count of collisions.

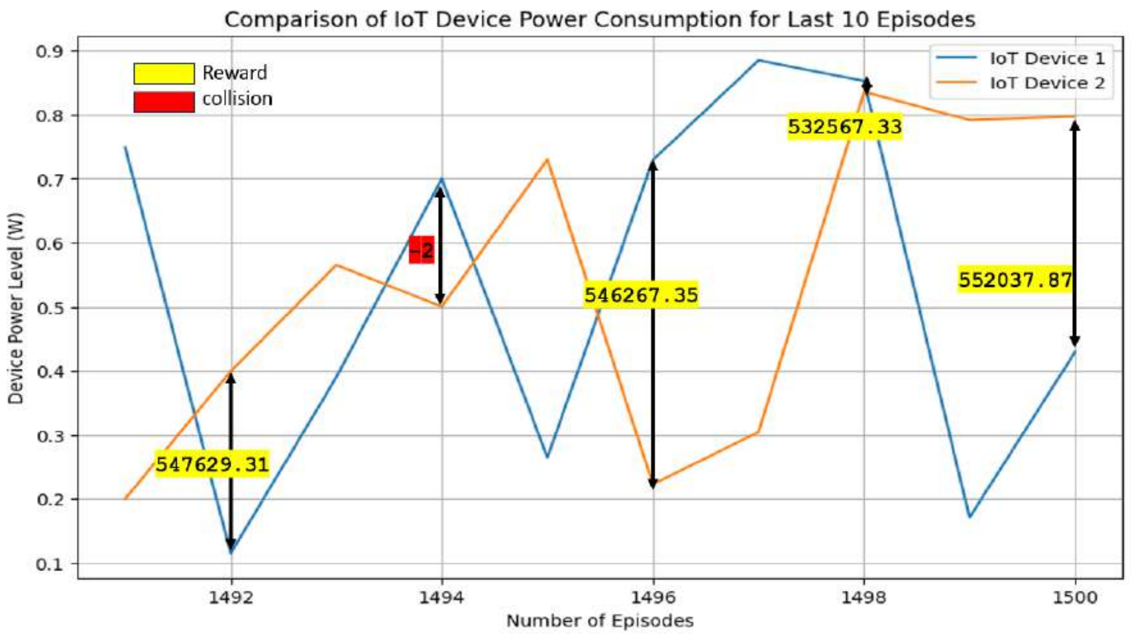

Figure 19: The plot illustrates the impact of changing batch sizes on the sum rate achieved by DQN Agents. After optimizing the sum rate with the best learning rate (0.001), this analysis explores the performance sensitivity to different batch sizes.

The above Fig. 19 shows the power consumption by both the IoT devices over the course of optimization for the last 10 episodes. During episode 1494, the sum rate achieved by the IoT devices was -2 bps, which indicates a collision. A collision in this context refers to an event where two or more devices attempt to transmit a signal at the same time over a shared communication channel, resulting in an overlap and interference of the signals. This collision disrupts the normal functioning of the devices and negatively impacts the data transmission. However, for the rest of the episodes, the sum rate was a specific positive value, indicating successful and collision-free data transmission between the devices. This fluctuation in sum rate underscores the importance of effective collision management in IoT devices to ensure optimal performance.

To conclude, lowering the learning rate from 0.1 to 0.001 in DQN training demonstrates a progressive reduction in collision rates over time, indicating that a slower learning rate facilitates more stable and consistent convergence towards improved performance in the later episodes.

4.3 Energy Efficiency Comparison

The energy efficiency comparison is depicted in Fig. 20. The energy efficiency metric is of important consideration especially in UAV assisted scenario. The energy efficiency is calculated by:

$$EE = \frac{\text{Sum Rate}}{\text{Power of IoT}}$$

Analyzing the impact of different learning rates on energy efficiency reveals distinct patterns in the behavior of the DQN algorithm. For a learning rate of 0.001, depicted by the green line, the energy efficiency steadily increases over episodes, reaching a peak slightly above 1.2e6 and close to 1.4e6. This gradual ascent indicates stable convergence.

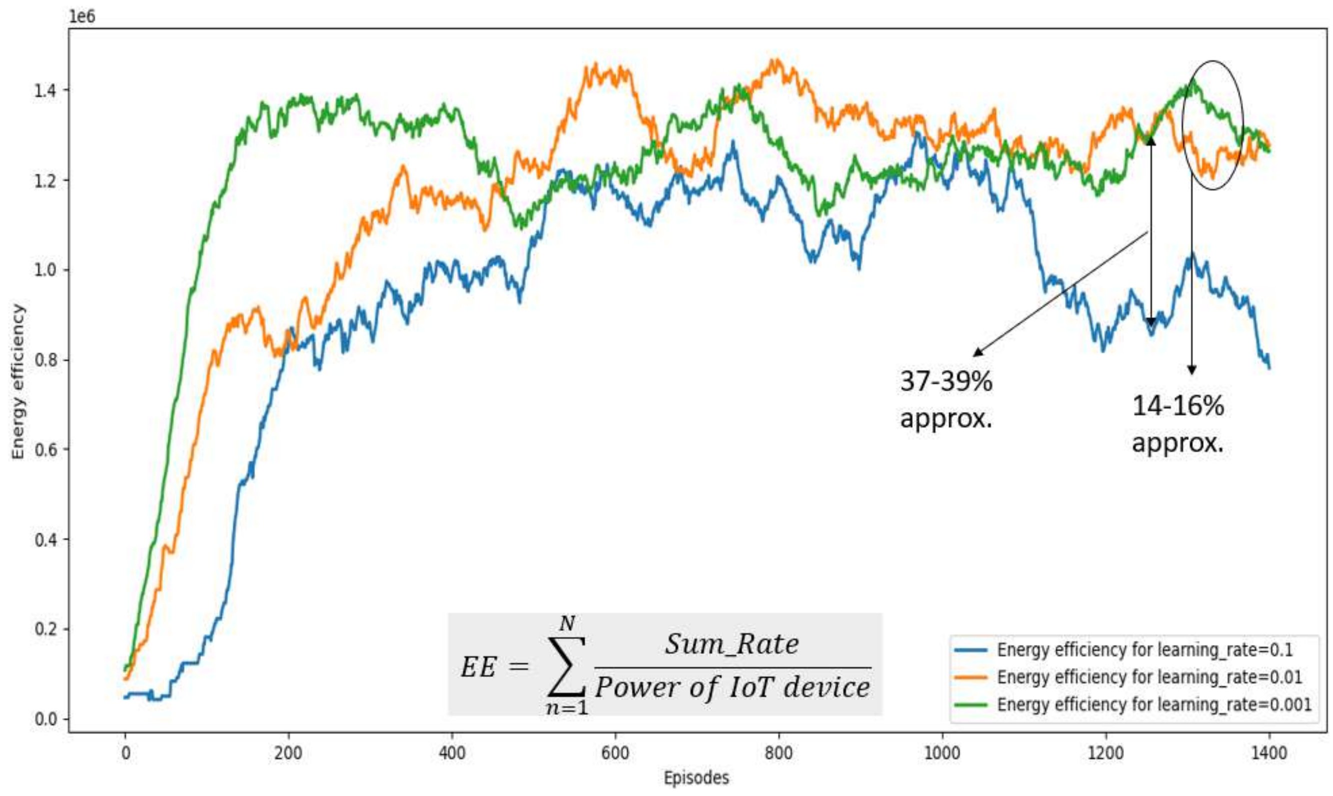

Figure 20: Average energy efficiency comparison with its varying learning rates of DQN.

In contrast, the orange line representing a learning rate of 0.01 exhibits a more aggressive behavior, resulting in rapid initial growth in energy efficiency and lies at a performance gap of **14-16%** approximately from 0.001 learning rate. However, around the 400-episode mark, oscillations become prominent, signaling potential challenges in stability. This could imply faster convergence but may come at the expense of adaptability, potentially leading to over fitting or difficulties in efficiently adapting to new data.

The blue line, corresponding to a learning rate of 0.1, showcases an almost immediate rise to peak efficiency but is accompanied by significant oscillations throughout the training process. These oscillations suggest potential instability and hint at the risk of over-fitting issues. As it lies at a huge performance gap of **37-39%** approximately from 0.001 learning rate which gives us another insight how the performance of our both agents(UAVs) has decreased over the course of episodes.

To sum up, our examination of various learning rates (0.001, 0.01, and 0.1) in the DQN algorithm highlights subtle trade-offs between how quickly the system converges and how stable that convergence is. With a learning rate of 0.001, we observe a steady and gradual increase in energy efficiency, suggesting stable convergence. Conversely, a learning rate of 0.01 shows rapid initial growth but introduces oscillations, indicating challenges in stability. The highest learning rate, 0.1, quickly reaches peak efficiency but exhibits significant oscillations, signaling potential instability issues. These findings emphasize the importance of selecting a learning rate tailored to specific needs through further exploration and fine-tuning.

4.4 Spectrum Efficiency Comparison

The spectrum efficiency (SE) of DQN agents trained with three different learning rates. The green line, representing a learning rate of 0.001, shows a steady and gradual increase in SE over episodes, reaching a plateau around episode 600. This indicates consistent performance with higher convergence and ultimately sets the benchmark for the other learning rates.

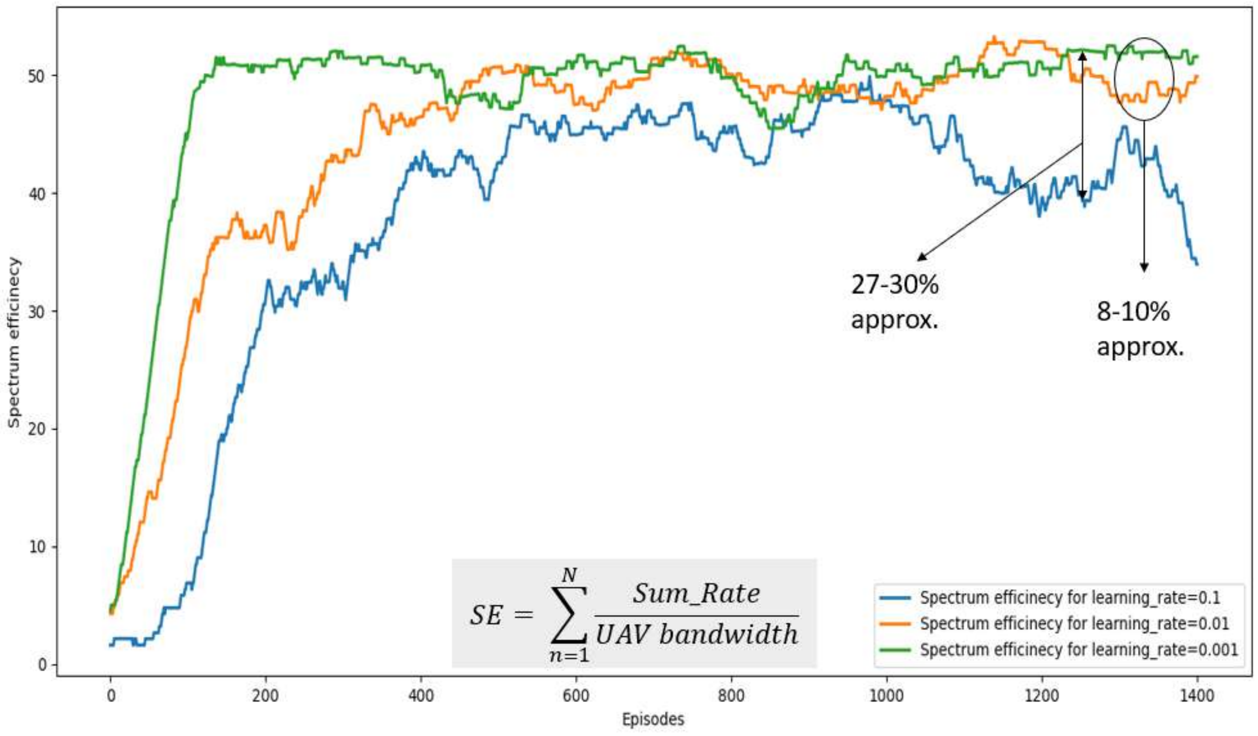

Figure 21: Average spectrum efficiency comparison with its varying learning rates of DQN.

In contrast, the orange line, representing a learning rate of 0.01, escalates rapidly in initial episodes but starts to fluctuate around episode 400 before stabilizing near episode 800. This could imply faster convergence but potential instability during training. Moreover, this lies at a performance gap of **8-10%** approximately.

The blue line, representing a learning rate of 0.1, shows rapid attainment of peak SE but has a huge performance gap of **27-30%** approximately which is Unfavorable to agents; moreover, it is followed by significant fluctuations throughout, suggesting potential over fitting or instability due to the aggressive update of Q-values.

In a nutshell, the learning rate of **0.001 (Green Line)** appears to achieve the best Spectrum Efficiency (SE) over the course of the episodes. It shows a steady and consistent increase in SE, indicating a stable learning process and effective utilization of the spectrum.

4.5 Deciphering DQN Learning Rates: A Loss Perspective

The plot below shows the loss calculated using Mean Square Error (MSE) for DQN agents trained with three different learning rates.

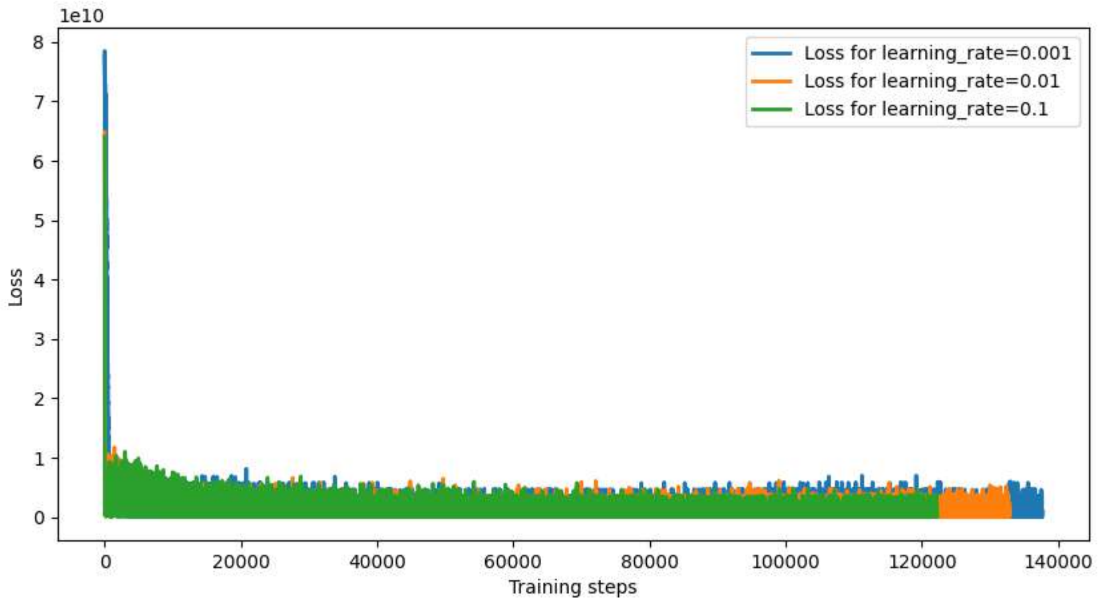

Figure 22: Comparative Loss Analysis for DQN Agents with Learning Rates of 0.001, 0.01, and 0.1

1. Learning Rate 0.001 (Blue Line): This learning rate starts with a very high loss, nearly reaching 8e10. However, it rapidly decreases and stabilizes around zero after approximately 20,000 training steps. This indicates that initially, the model with this learning rate had a significant error but adjusted quickly.

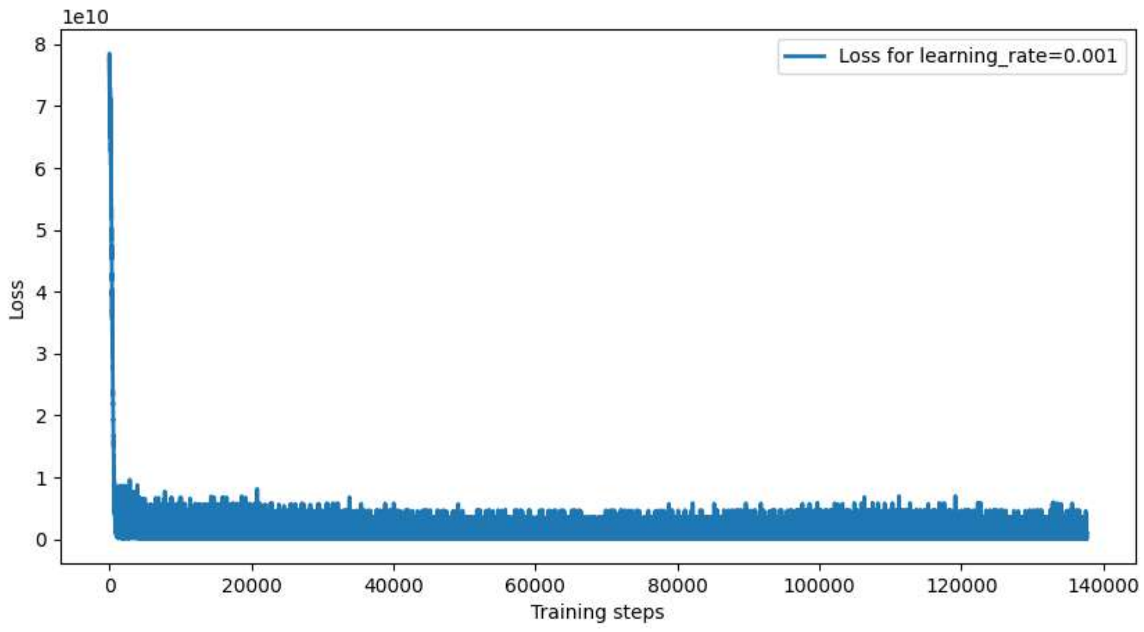

Figure 23: Loss for DQN Agents with Learning Rate of 0.001

2. Learning Rate 0.01 (Orange Line): The loss with this learning rate is significantly lower initially compared to the blue line. It also decreases rapidly and appears to stabilize close to zero, though it takes slightly more training steps to stabilize compared to the blue line.

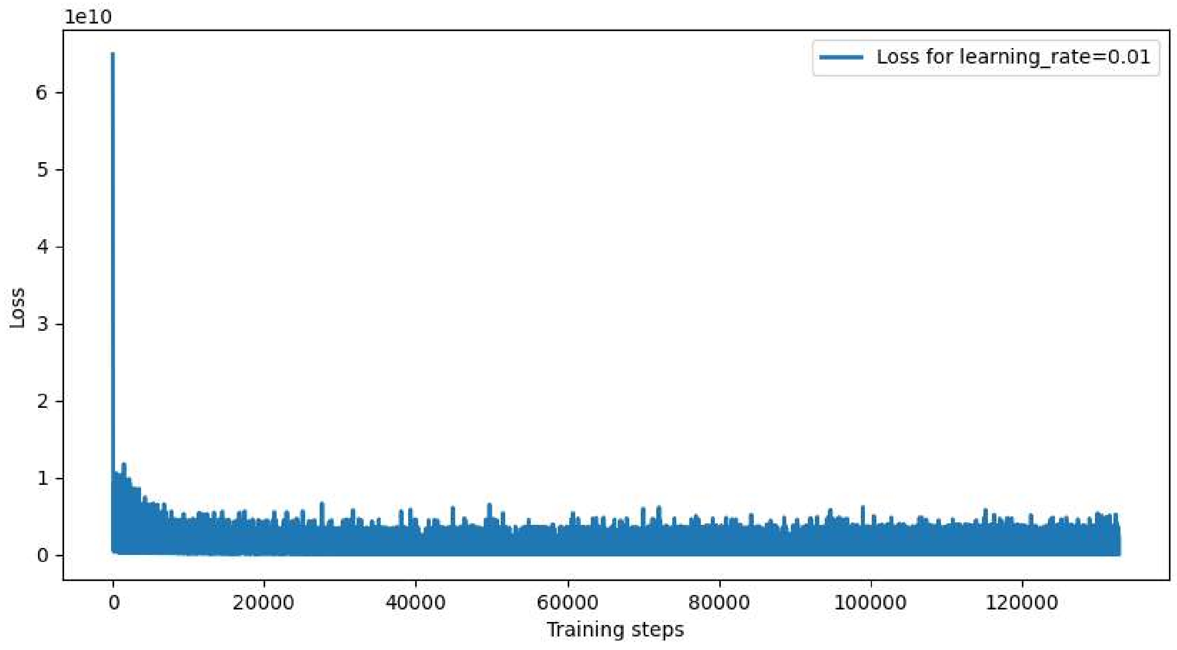

Figure 24: Loss for DQN Agents with Learning Rate of 0.01

3. Learning Rate 0.1 (Green Line): This learning rate shows the lowest initial loss among all three. It remains relatively stable throughout the training process, indicating that this learning rate might be too high causing the model not to converge effectively or it has reached an optimal state quickly.

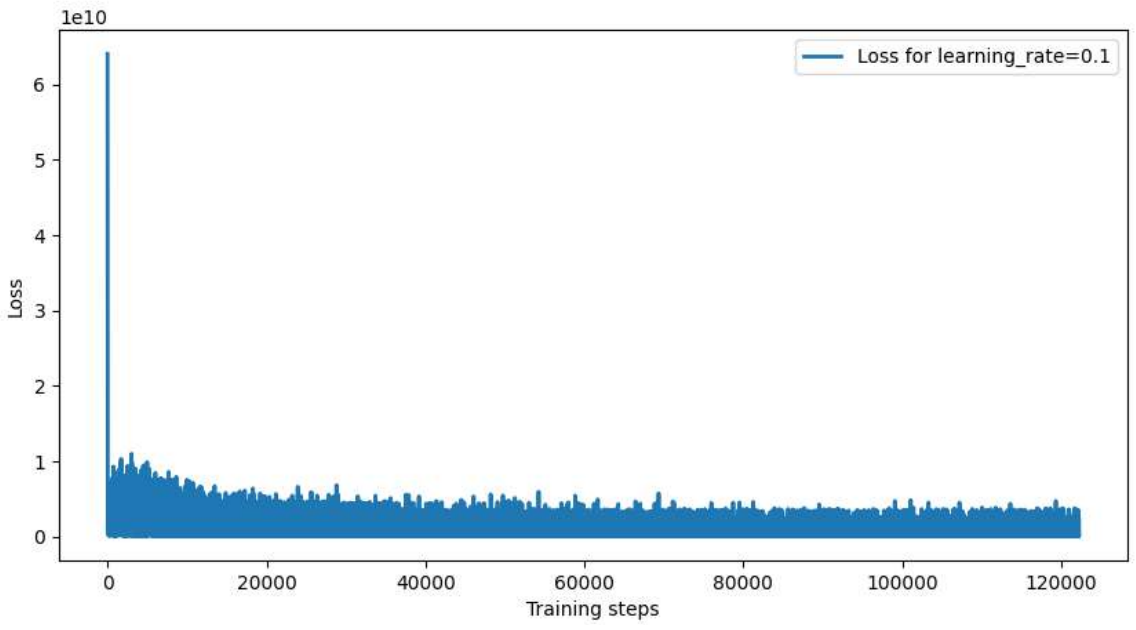

Figure 25: Loss for DQN Agents with Learning Rate of 0.1

In conclusion, A higher learning rate, while offering rapid initial learning, may lead to instability in the training process. On the other hand, a smaller learning rate, despite slower initial learning, tends to provide more stable and consistent performance over a larger number of episodes.

However, it's crucial to note that these observations are specific to the conditions of this particular experiment. The optimal learning rate may vary under different scenarios or parameters. This approach ensures robust and efficient training of DQN agents, ultimately leading to improved performance in UAV-assisted IoT networks.

4.6 DQN Batch Size Analysis

Following the successful implementation of DQN algorithms with various learning rates, the total data rate (sum rate) was optimized. The optimal learning rate of 0.001 was chosen and subsequently examined with different batch sizes to assess the impact of varying batch size on DQN Agents' performance.

The resulting Fig. 26 illustrates the achieved sum rate across different batch sizes for the selected optimal learning rate of 0.001.

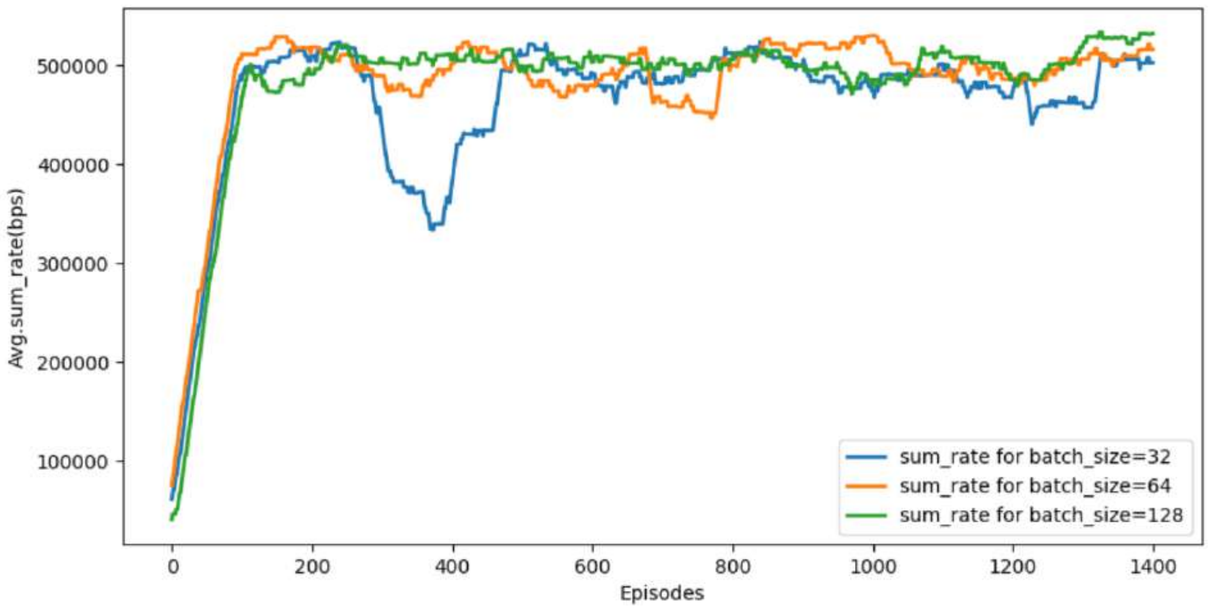

Figure 26: The plot illustrates the impact of changing batch sizes on the sum rate achieved by DQN Agents. After optimizing the sum rate with the best learning rate (0.001), this analysis explores the performance sensitivity to different batch sizes.

1. **Batch Size 32 (Blue Line)**: Initially, the DQN agent with a batch size of 32 exhibits a swift increase in the average sum rate, though a noticeable dip occurs around the 400-episode mark. Despite this dip, performance recovers and stabilizes, aligning with the other batch sizes.

2. **Batch Size 64 (Orange Line)**: The DQN agent with a batch size of 64 also demonstrates a sharp performance increase initially. Unlike the batch size of 32, this

line doesn't experience a significant dip and maintains relatively stable performance throughout the episodes.

3. Batch Size 128 (Green Line): The DQN agent's performance with a batch size of 128 follows a similar trend to the batch size of 64, showing a consistent increase in the average sum rate and maintaining stability throughout the episodes.

Overall, the effect of batch size seems to be insensitive to DQN performance as all three batch sizes exhibit similar performance trends over the episodes. Despite some fluctuations, all three lines stabilize towards the end of the observed episodes, indicating that the DQN agent is able to learn effectively regardless of the batch size.

Chapter 5

5 Conclusion and Future Direction

In conclusion, the analysis of DQN agents with varying learning rates in the context of Multi-Agent Reinforcement Learning for sum rate maximization in UAV-assisted IoT networks has yielded insightful results. The consistent outperformance of lower learning rates (0.001) in metrics such as sum rate, energy efficiency, and spectral efficiency underscores the significance of meticulous learning rate selection. Notably, the observed reduction in collisions enhances the safety aspect of the system. However, the trade-off between convergence speed and stability is evident, as lower learning rates exhibit faster convergence, as seen in the loss plots.

Furthermore, different reinforcement learning (RL) approaches have been developed to address the challenges encountered in deploying UAVs to support IoT networks. These frameworks are designed with a focus on performance metrics specific to the IoT environment, where UAVs play a crucial role. Drawing from the findings outlined in this report, potential future research directions may include:

As this study opens avenues for further exploration. The applicability of alternative Deep Reinforcement Learning (DRL) algorithms, such as Q-learning, Deep Deterministic Policy Gradient (DDPG), and Proximal Policy Optimization (PPO), warrants investigation to assess their impact on performance. Additionally, experimenting with different action settings, such as adjusting the angle of rotation or velocity of UAVs, could provide valuable insights into optimizing resource utilization and system efficiency [33].

The titled ”Multi-Agent Reinforcement Learning for Sum Rate Maximization in UAV-Assisted IoT Networks,” employing two UAVs and two IoT devices, offers a foundational understanding. To enhance its applicability, future endeavors could explore scaling up the system by increasing the number of UAVs and IoT devices. Real-time considerations and dynamic scenarios may further enrich the model's robustness and practicality [33, 34, 35].

Building upon the RL strategies introduced in this report the envisioned future direction involves the hardware realization of these methods in on-board and hardware-in-loop configurations to support UAV-enabled IoT and heterogeneous networks. Integrating RL frameworks for on-board learning introduces novel avenues and dimensions for further exploration and research [34]. The process of identifying, designing, programming, and integrating hardware with the aforementioned RL frameworks necessitates careful consideration of complexity and energy consumption aspects. Evaluating the performance trade-offs between hardware energy consumption, algorithmic complexity, and convergence potential offers valuable insights [35]. Through thorough analysis and investigation, it becomes possible to develop efficient hardware designs tailored specifically for RL frameworks, thus enhancing their effectiveness in real-world applications.

In the broader context, the integration of Unmanned Aerial Vehicles (UAVs) into communication services for IoT demonstrates a promising trajectory. Leveraging the synergy of UAVs, IoT, and Artificial Intelligence (AI) presents a compelling avenue for innovation. As UAVs evolve into intelligent, self-directed entities, the potential for delivering enhanced services grows exponentially. This report provides a comprehensive overview of UAV communication, IoT, and RL technologies, addressing potential challenges, applications, and developmental trajectories within the evolving landscape of UAV-assisted IoT.

Appendices

A Program Code

A.1 Importing Libraries

```python

import gymnasium as gym
import numpy as np
import tree
from ray.rllib.env.multi_agent_env import MultiAgentEnv
import torch
import torch.nn.functional as F
import torch.nn as nn
import torch.optim as optim
from torch.autograd import Variable
import random
import matplotlib.pyplot as plt
```

A.2 Environment Modelling

```python

class CustomUAVEnv(MultiAgentEnv):
    def __init__(self, config = None):
        super().__init__()
        self.grid_size = 10
        self.num_uavs = 2
        self.num_devices = 2
        self._agent_ids = {'agent_0', 'agent_1'}#, 'agent_2'}

        # Initialize grid and entities
        self.grid = np.zeros((self.grid_size, self.grid_size), dtype=float)
        self.uav_positions = np.random.randint(0, self.grid_size, size=(self.num_uavs,
        ↪  2))
        self.device_positions = np.random.randint(0, self.grid_size,
        ↪  size=(self.num_devices, 2))
        self.device_powers = 0.1+0.9*np.random.rand(self.num_devices) #device power
        ↪  between 100 mW and 1 watts
        self.uav_bandwidths = np.full(self.num_uavs, 10000) #UAV bandiwdth is 1MHz
        self.noise = 1e-12 # -90dBm
        self.uav_height = 10 #10m height
        self.steps_in_traj = 0
        # Define observation and action spaces
        self.observation_space = gym.spaces.Box(low=0, high=500, shape=(8, ),
        ↪  dtype=np.float32)
        self.action_space = gym.spaces.Discrete(5)  # Assuming 4 discrete actions (e.g.,
        ↪  no movement, up, down, left, right)
        self.random_state = np.random.RandomState()
        self.infos = {"agent_0":{"collisions": 0},"agent_1":{"collisions": 0}}#,
        ↪  "agent_2":{"collisions": 0}}

    def reset(self):
        # Reset environment state
        self.uav_positions = np.random.randint(0, self.grid_size, size=(self.num_uavs,
        ↪  2))
        self.device_positions = np.random.randint(0, self.grid_size,
        ↪  size=(self.num_devices, 2))
        self.device_powers = 0.1+0.9*np.random.rand(self.num_devices) #device power
        ↪  between 100 mW and 1 watts
        self.uav_bandwidths = np.full(self.num_uavs, 10000) #UAV bandiwdth is 1MHz
        self.steps_in_traj = 0
        self.infos = {"agent_0":{"collisions": 0},"agent_1":{"collisions": 0}}#,
        ↪  "agent_2":{"collisions": 0}}

        # Construct observations for each agent
        observations = {}
        for i in range(self.num_uavs):
            obs = self._construct_observation(i)
            observations[f"agent_{i}"] = obs
        return observations

    def step(self, actions):
        self.steps_in_traj += 1

        # Update UAV positions based on actions
        for i, action in enumerate(actions.values()):
            if action == 0: # do not move the UAV
              pass
            elif action == 1:# move UAV up
```

```
                self.uav_positions[i][0] += 1
            elif action == 2: # move UAV down
                self.uav_positions[i][0] -= 1
            elif action == 3: # move UAV left
                self.uav_positions[i][1] -= 1
            elif action == 4: # move UAV right
                self.uav_positions[i][1] += 1

        if (self.uav_positions[i][0] < 0) | (self.uav_positions[i][1] < 0) |
        ↪  (self.uav_positions[i][0] >= self.grid_size) | (self.uav_positions[i][1] >=
        ↪  self.grid_size):
          terminateds = {"__all__": True, "agent_0": True, "agent_1": True}
          rewards = {}
          for i in range(self.num_uavs):
            rewards[f"agent_{i}"] = -1
          # Construct next observations
          next_observations = {}
          for i in range(self.num_uavs):
            next_obs = self._construct_observation(i)
            next_observations[f"agent_{i}"] = next_obs
          return next_observations, rewards, terminateds, self.infos

        # Calculate rewards for each agent
        rewards = {}
        for i in range(self.num_uavs):
            reward = self._calculate_reward(i)
            rewards[f"agent_{i}"] = reward

        # Construct next observations
        next_observations = {}
        for i in range(self.num_uavs):
            next_obs = self._construct_observation(i)
            next_observations[f"agent_{i}"] = next_obs

        if self.steps_in_traj > 100:
            terminateds = {"__all__": True, "agent_0": True, "agent_1": True}#,
            ↪  "agent_2": True}
        else:
            terminateds = {"__all__": False, "agent_0": False, "agent_1": False}#,
            ↪  "agent_2": False}

        return next_observations, rewards, terminateds, self.infos

    def _construct_observation(self, agent_id):

        obs = [self.uav_bandwidths[agent_id]]

        # position of other UAVs and bandwidths
        for i in range(self.num_uavs):
            if i != agent_id:
                obs.append(self.uav_bandwidths[i])

        # position of ground devices (1, 0) and power
        for i in range(self.num_devices):
            obs.append(self.device_powers[i])

        return np.concatenate([self.uav_positions[0], self.uav_positions[1], obs],
        ↪  dtype=np.float16)

    def _calculate_reward(self, agent_id):

        collision = any(
            (i != agent_id) and np.all(self.uav_positions[i] ==
            ↪  self.uav_positions[agent_id])
            for i in range(self.num_uavs)
            )

        if collision:
          self.infos[f'agent_{agent_id}']['collisions'] += 1
          return -1

        # find distance from the assigned device
        device_dists = self._calculate_gn(self.uav_positions[agent_id], self.uav_height,
        ↪  self.device_positions[agent_id])

        alpha_n = 0.1
        sum_sinr = alpha_n*self.device_powers[agent_id]*device_dists/self.noise

        data_rate = self.uav_bandwidths[agent_id]*np.log2(1 + sum_sinr)/1

```

```python
        return data_rate

    def seed(self, seed=None):
        if seed is None:
            seed = np.random.randint(2**31)
        self.random_state.seed(seed)

    def _calculate_gn(self, uav_location, uav_height, device_location):
        return 1/np.sqrt((uav_location[0] - device_location[0])**2 + (uav_location[1] -
        ↪  device_location[1])**2 + uav_height**2)**2.3

    def render(self, mode='human'):
        img = np.zeros((self.grid_size, self.grid_size, 3), dtype=np.uint8)

        # Render UAVs as red circles
        for pos in self.uav_positions:
            img[pos[0], pos[1]] = [255, 0, 0]

        # Render devices as blue squares
        for pos in self.device_positions:
            img[pos[0], pos[1]] = [0, 0, 255]

        plt.imshow(img)
        plt.show()
        return img

```

A.3 DQN AGENT: Agent Network

```python
class QNetwork(nn.Module):
    def __init__(self, state_shape, num_actions):
        super(QNetwork, self).__init__()
        self.fc1 = nn.Linear(state_shape, 32)
        self.fc2 = nn.Linear(32, num_actions)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        return self.fc2(x)
```

A.4 DQN AGENT: Network Code

```python
class DQNAgent:
    def __init__(self, state_shape, num_actions, batch_size=64, lr=0.001, replay_buffer_size=1000, gamma=0.99):
        self.state_shape = state_shape
        self.num_actions = num_actions
        self.model = QNetwork(state_shape, num_actions)
        self.target_model = QNetwork(state_shape, num_actions)
        self.learning_rate = lr
        self.optimizer = optim.Adam(self.model.parameters(), lr= self.learning_rate)
        self.loss_fn = nn.MSELoss()
        self.gamma = gamma  # Discount factor
        self.epsilon = 1.0  # Exploration-exploitation trade-off
        self.epsilon_min = 0.01
        self.explore_step = 4000
        self.epsilon_decay = (self.epsilon - self.epsilon_min) / self.explore_step
        self.batch_size = batch_size
        self.replay_buffer = []  # Initialize replay buffer
        self.replay_buffer_size = replay_buffer_size
        self.tau = 0.01
        self.losses = []  # New variable to store losses

        # hard copy model parameters to target model parameters
        for target_param, param in zip(self.model.parameters(), self.target_model.parameters()):
            target_param.data.copy_(param)

    def soft_update(self):
        """Soft update model parameters.
        _target = *_local + (1 - )*_target
        """
        for target_param, local_param in zip(self.target_model.parameters(), self.model.parameters()):
            target_param.data.copy_(self.tau*local_param.data + (1.0-self.tau)*target_param.data)

    def act(self, state):
        if np.random.rand() <= self.epsilon:
            return np.random.choice(self.num_actions)
        state = torch.FloatTensor(state).unsqueeze(0)
        q_values = self.model(state).detach().numpy()
        return np.argmax(q_values)

    def train(self):

        # Sample a mini-batch from the replay buffer
        mini_batch = random.sample(self.replay_buffer, self.batch_size)

        # Prepare the mini-batch
        states, actions, rewards, next_states, dones = zip(*mini_batch)
        states = torch.FloatTensor(np.array(states))
        actions = torch.LongTensor(np.array(actions))
        rewards = torch.FloatTensor(np.array(rewards))
        next_states = torch.FloatTensor(np.array(next_states))
        dones = torch.FloatTensor(np.array(dones))

        # Compute targets for the mini-batch
        q_values = self.model(states)
        next_q_values = self.target_model(states)
        q_values = q_values.gather(1, actions.unsqueeze(1))
        expected_q_values = rewards + self.gamma * (1 - dones) *
        ↪  next_q_values.detach().max(1)[0]
        expected_q_values = expected_q_values.unsqueeze(1)
        # Perform a single training step on the mini-batch
        loss = self.loss_fn(q_values, expected_q_values)  # Ensure shapes match
        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()

        # Store the loss in the list
        self.losses.append(loss.item())

        if self.epsilon > self.epsilon_min:
          self.epsilon -= self.epsilon_decay

```

```python
    def update_target_model(self):
        self.target_model.set_weights(self.model.get_weights())

    def save_model(self, filename):
      torch.save(self.model.state_dict(), filename)

    def load_model(self, filename):
      self.model.load_state_dict(torch.load(filename, map_location=torch.device('cpu')))
      self.model.eval()

```

A.5 DQN AGENT: Training Procedure

```python
def train(env, learning_rate, num_episodes=1500, gamma=0.99):
    print(f'Training for learning rate: {learning_rate} .......')
    # Create DQNAgent instances for each UAV
    agent_0 = DQNAgent(state_size, num_actions, lr=learning_rate, gamma=gamma)
    agent_1 = DQNAgent(state_size, num_actions, lr=learning_rate, gamma=gamma)

    steps = 0
    scores_0 = []
    scores_1 = []
    device_powers_0 = []
    uav_bandwidths_0 = []
    device_powers_1 = []
    uav_bandwidths_1 = []
    best_score_0 = 0
    best_score_1 = 0

    for episode in range(num_episodes):
        state = env.reset()

        done = {"__all__": False, "agent_0": False, "agent_1": False}

        while not done["__all__"]:

            action_0 = agent_0.act(state['agent_0'])
            action_1 = agent_1.act(state['agent_1'])
            next_state, rewards, done, _ = env.step({'agent_0': action_0, 'agent_1': action_1})

            agent_0.replay_buffer.append((state['agent_0'], action_0, rewards['agent_0'], next_state['agent_0'], done['agent_0']))
            agent_1.replay_buffer.append((state['agent_1'], action_1, rewards['agent_1'], next_state['agent_1'], done['agent_1']))

            state = next_state
            steps += 1

            if steps % 100 and len(agent_0.replay_buffer) >= 64 and len(agent_1.replay_buffer) >= 64:
              agent_0.train()
              agent_1.train()

        print("Episode number: {}, Sum_rate(bps): {}".format(episode, round(sum(rewards.values()), 2)))

        scores_0.append(rewards['agent_0'])
        scores_1.append(rewards['agent_1'])
        device_powers_0.append(env.device_powers[0])
        device_powers_1.append(env.device_powers[1])
        uav_bandwidths_0.append(env.uav_bandwidths[0])
        uav_bandwidths_1.append(env.uav_bandwidths[1])
        avg_score_0 = np.mean(scores_0[-min(100, len(scores_0)):])
        avg_score_1 = np.mean(scores_1[-min(100, len(scores_1)):])

        if avg_score_0 > best_score_0:
          agent_0.save_model(f'the_best_agent0_{learning_rate}.pth')
          best_score_0 = avg_score_0

        if avg_score_1 > best_score_1:
          agent_1.save_model(f'the_best_agent1_{learning_rate}.pth')
          best_score_1 = avg_score_1

        if episode % target_update_interval == 0:
            agent_0.soft_update()
            agent_1.soft_update()
```

```python

    # Save rewards to files
    np.save(f'rewards_agent_0_{learning_rate}.npy', np.array(scores_0))
    np.save(f'rewards_agent_1_{learning_rate}.npy', np.array(scores_1))
    np.save(f"agent_0_losses_lr_{learning_rate}.npy", np.array(agent_0.losses))
    np.save(f"agent_1_losses_lr_{learning_rate}.npy", np.array(agent_1.losses))
    np.save(f'device_powers_0_{learning_rate}.npy', np.array(device_powers_0))
    np.save(f'device_powers_1_{learning_rate}.npy', np.array(device_powers_1))
    np.save(f"uav_bandwidths_0_{learning_rate}.npy", np.array(uav_bandwidths_0))
    np.save(f"uav_bandwidths_1_{learning_rate}.npy", np.array(uav_bandwidths_1))


def start_training(lr1=0.1,lr2=0.01,lr3=0.001, episodes=1500, gamma=0.99):
    if env is None or num_actions is None or state_size is None:
        initialize_globals()
    for lr in [lr1, lr2, lr3]:
        # Train
        train(env, lr, episodes, gamma)

```

A.6 Plotting Reward Curves

```python

def load_rewards(agent_id, learning_rate):
    rewards= np.load(f'rewards_agent_{agent_id}_{learning_rate}.npy')
    device_power = np.load(f'device_powers_{agent_id}_{learning_rate}.npy')
    bandwidth = np.load(f'uav_bandwidths_{agent_id}_{learning_rate}.npy')
    return rewards, device_power, bandwidth


def moving_average(data, window_size): # Calculate the average of the sum-rate on each
↪   100 episodes to verify if the agents learn (increasing curve) or not (fluctuations)
    return np.convolve(data, np.ones(window_size)/window_size, mode='valid')


def plot_sum_rate(window_size, lr1, lr2, lr3):

    plt.figure(figsize=(15, 7))
    for learning_rate in [lr1, lr2, lr3]:
        rewards0, _, _ = load_rewards(agent_id=0, learning_rate=learning_rate)
        rewards1, _, _ = load_rewards(agent_id=1, learning_rate=learning_rate)

        moving_avg = moving_average(rewards0 + rewards1, window_size)

        plt.plot(moving_avg, label=f'sum_rate for learning_rate={learning_rate}',
         ↪  linewidth=2)

    # Adding labels and title
    plt.xlabel('Episodes')
    plt.ylabel(' Avg. Sum_rate (bps)')
    plt.legend()

    svg_file = 'sum_rate.svg'
    plt.savefig(svg_file, format='svg')

    plt.show()

```

A.7 Plotting Loss Function (for Agent-0 for example)

```


    def load_loss(agent_id, learning_rate):
        return np.load(f'agent_{agent_id}_losses_lr_{learning_rate}.npy')

    def plot_loss(loss, learning_rate):
        plt.plot(loss, label=f'Loss for learning_rate={learning_rate}', linewidth=2)

    # Load loss from files (of the agent_0)
    loss0_001, loss0_01, loss0_1 = load_loss(0, lr3), load_loss(0, lr2), load_loss(0,
     ↪  lr1)

    plt.figure(figsize=(10, 5))

    # Plotting the Loss
    #plot_loss(loss0_001, 0.001)
```

```
#plot_loss(loss0_01, 0.01)
plot_loss(loss0_1, lr1)

# Adding labels and title
plt.xlabel('Training steps')
plt.ylabel('Loss')
plt.legend()

# Save the plot as an SVG file
plt.savefig('Loss_plot.svg', format='svg')
# Display the plot
plt.show()

def load_loss(agent_id, learning_rate):
    return np.load(f'agent_{agent_id}_losses_lr_{learning_rate}.npy')

def plot_loss(loss, learning_rate):
    plt.plot(loss, label=f'Loss for learning_rate={learning_rate}', linewidth=2)

# Load loss from files (of the agent_0)
loss0_001, loss0_01, loss0_1 = load_loss(0, lr3), load_loss(0, lr2), load_loss(0,
↪  lr1)

plt.figure(figsize=(10, 5))

# Plotting the Loss
#plot_loss(loss0_001, 0.001)
plot_loss(loss0_01, lr2)
#plot_loss(loss0_1, 0.1)

# Adding labels and title
plt.xlabel('Training steps')
plt.ylabel('Loss')
plt.legend()

# Save the plot as an SVG file
plt.savefig('Loss_plot.svg', format='svg')
# Display the plot
plt.show()

```

```
def load_loss(agent_id, learning_rate):
    return np.load(f'agent_{agent_id}_losses_lr_{learning_rate}.npy')

def plot_loss(loss, learning_rate):
    plt.plot(loss, label=f'Loss for learning_rate={learning_rate}', linewidth=2)

# Load loss from files (of the agent_0)
loss0_001, loss0_01, loss0_1 = load_loss(0, lr3), load_loss(0, lr2), load_loss(0,
↪  lr1)

plt.figure(figsize=(10, 5))

# Plotting the Loss
plot_loss(loss0_001, lr3)
#plot_loss(loss0_01, 0.01)
#plot_loss(loss0_1, 0.1)

# Adding labels and title
plt.xlabel('Training steps')
plt.ylabel('Loss')
plt.legend()

# Save the plot as an SVG file
plt.savefig('Loss_plot.svg', format='svg')
# Display the plot
plt.show()

"""**ALL PLOTS OF LOSSES**"""

def load_loss(agent_id, learning_rate):
    return np.load(f'agent_{agent_id}_losses_lr_{learning_rate}.npy')

def plot_loss(loss, learning_rate):
    plt.plot(loss, label=f'Loss for learning_rate={learning_rate}', linewidth=2)

# Load loss from files (of the agent_0)
loss0_001, loss0_01, loss0_1 = load_loss(0, lr3), load_loss(0, lr2), load_loss(0,
↪  lr1)

plt.figure(figsize=(10, 5))
```

```

    # Plotting the Loss
    plot_loss(loss0_001, lr3)
    plot_loss(loss0_01, lr2)
    plot_loss(loss0_1, lr1)

    # Adding labels and title
    plt.xlabel('Training steps')
    plt.ylabel('Loss')
    plt.legend()

    # Save the plot as an SVG file
    plt.savefig('Loss_plot.svg', format='svg')
    # Display the plot
    plt.show()

```

A.8 Plotting Cumulative Sum Rate

```


    import matplotlib.pyplot as plt
    import numpy as np

    plt.figure(figsize=(12, 7))

    for learning_rate in [lr1,lr2,lr3]:
        rewards0, _, _ = load_rewards(agent_id=0, learning_rate=learning_rate)
        rewards1, _, _ = load_rewards(agent_id=1, learning_rate=learning_rate)

        sum_rates = rewards0 + rewards1
        max_sum_rate = np.max(sum_rates)

        plt.bar(str(learning_rate), max_sum_rate, label=f'Max Sum Rate for Learning Rate
         ↪  = {learning_rate}')

    # Adding labels and title<
    plt.xlabel('Episodes')
    plt.ylabel('Max Sum Rate (bps)')
    plt.legend()

    # Set y-axis limit to start from 500,000
    plt.ylim(500000, plt.ylim()[1])

    # Save the plot as an SVG file
    plt.savefig('max_sum_rate.svg', format='svg')
    # Display the plot
    plt.show()

```

A.9 Plotting Energy Eficiency

```

    plt.figure(figsize=(15, 7))
    for learning_rate in [lr1,lr2,lr3]:
        rewards0, device_power0, _ = load_rewards(agent_id=0,
         ↪ learning_rate=learning_rate)
        rewards1, device_power1, _ = load_rewards(agent_id=1,
         ↪ learning_rate=learning_rate)

        # Calculate the ratio of rewards to device_power
        ratio0 = rewards0 / device_power0
        ratio1 = rewards1 / device_power1

        moving_avg0 = moving_average(ratio0, window_size)
        moving_avg1 = moving_average(ratio1, window_size)

        plt.plot(moving_avg0 + moving_avg1, label=f'Energy efficiency for
         ↪ learning_rate={learning_rate}', linewidth=2)

    # Adding labels and title
    plt.xlabel('Episodes')
    plt.ylabel('Energy efficiency')
    plt.legend()

    # Save the plot as an SVG file
    plt.savefig('Energy_efficiency.svg', format='svg')
    # Display the plot
    plt.show()

```

A.10 Plotting Spectrum Eficiency

```

    #Spectrum efficinecy = (sum_rate in each episode) / (total bandwidth of UAVs)
    plt.figure(figsize=(15, 7))
    for learning_rate in [lr1,lr2,lr3]:
        rewards0, _, bandwidth0 = load_rewards(agent_id=0, learning_rate=learning_rate)
        rewards1, _, bandwidth1 = load_rewards(agent_id=1, learning_rate=learning_rate)

        # Calculate the ratio of rewards to bandwidth
        ratio0 = rewards0 / bandwidth0
        ratio1 = rewards1 / bandwidth1

        moving_avg = moving_average(ratio0 + ratio1, window_size)

        plt.plot(moving_avg, label=f'Spectrum efficinecy for
         ↪  learning_rate={learning_rate}', linewidth=2)

    # Adding labels and title
    plt.xlabel('Episodes')
    plt.ylabel('Spectrum efficinecy')
    plt.legend()

    # Save the plot as an SVG file
    plt.savefig('Spectrum_efficinecy.svg', format='svg')
    # Display the plot
    plt.show()

```

A.11 Plotting Collision Rate

```python

    import numpy as np
    import matplotlib.pyplot as plt

    plt.figure(figsize=(15, 7))

    for learning_rate in [lr1,lr2,lr3]:
        rewards0, _, _ = load_rewards(agent_id=0, learning_rate=learning_rate)
        rewards1, _, _ = load_rewards(agent_id=1, learning_rate=learning_rate)
        sum_rate = rewards0 + rewards1

        # Count occurrences of -2 in the sum_rate array
        collision_count = np.count_nonzero(sum_rate == -2)

        plt.bar(str(learning_rate), collision_count, label=f'Collisions for Learning
         ↪  Rate = {learning_rate}')

    # Adding labels and title
    plt.xlabel('Episodes')
    plt.ylabel(' Number of Collisions')
    plt.legend()

    # Save the plot as an SVG file
    plt.savefig('collision.svg', format='svg')
    # Display the plot
    plt.show()

```

A.12 Analyzing Different Batch-Sizes With Best Learning Rate 0.001

```python

import matplotlib.pyplot as plt

def load_rewards(agent_id, batch_size):
    return np.load(f'rewards_agent_{agent_id}_{batch_size}.npy')
```

```python

def moving_average(data, window_size):
    return np.convolve(data, np.ones(window_size)/window_size, mode='valid')

def plot_rate_sum(moving_avg1, moving_avg2, batch_size):
    plt.plot(moving_avg1 + moving_avg2, label=f'sum_rate for batch_size={batch_size}',
     ↪  linewidth=2)

# Load rewards from files
rewards0_32, rewards1_32 = load_rewards(0, 32), load_rewards(1, 32)
rewards0_64, rewards1_64 = load_rewards(0, 64), load_rewards(1, 64)
rewards0_128, rewards1_128 = load_rewards(0, 128), load_rewards(1, 128)

window_size = 100
moving_avg1_32, moving_avg2_32 = moving_average(rewards0_32, window_size),
 ↪  moving_average(rewards1_32, window_size)
moving_avg1_64, moving_avg2_64 = moving_average(rewards0_64, window_size),
 ↪  moving_average(rewards1_64, window_size)
moving_avg1_128, moving_avg2_128 = moving_average(rewards0_128, window_size),
 ↪  moving_average(rewards1_128, window_size)

plt.figure(figsize=(10, 5))

# Plotting the Rate Sum
plot_rate_sum(moving_avg1_32, moving_avg2_32, 32)
plot_rate_sum(moving_avg1_64, moving_avg2_64, 64)
plot_rate_sum(moving_avg1_128, moving_avg2_128, 128)

# Adding labels and title
plt.xlabel('Episodes')
plt.ylabel('Avg.sum_rate(bps)')
plt.legend()

# Save the plot as an SVG file
plt.savefig('sum_rate.svg', format='svg')
# Display the plot
plt.show()

```

B Bibliography

Bibliography

[1] Khurram Mahmud, Syed; Chen, Yue; Chai, Kok Keong (2022): Tandem RL Framework for Sum Rate Enhancement in NOMAUAV Network. TechRxiv. Preprint. Available at: https://doi.org/10.36227/techrxiv.20406972.v1

[2] W. Chen, J. Liu, H. Guo, and N. Kato, "Toward Robust and Intelligent Drone Swarm: Challenges and Future Directions," IEEE Network, vol. 34, no. 4, pp. 278–283, 2020.

[3] D. C. Nguyen, M. Ding, P. N. Pathirana, A. Seneviratne, J. Li, D. Niyato, O. Dobre, and H. V. Poor, "6G Internet of Things: A Comprehensive Survey," IEEE Internet of Things Journal, vol. 9, no. 1, pp. 359–383, 2022.

[4] B. Mao, F. Tang, Y. Kawamoto, and N. Kato, "AI Models for Green Communications Towards 6G," IEEE Communications Surveys and Tutorials, vol. 24, no. 1, pp. 210–247, 2022.

[5] P. McEnroe, S. Wang, and M. Liyanage, "A Survey on the Convergence of Edge Computing and AI for UAVs: Opportunities and Challenges," IEEE Internet of Things Journal, 2022.

[6] M.-A. Lahmeri, M. A. Kishk, and M.-S. Alouini, “Artificial Intelligence for UAV-Enabled Wireless Networks: A Survey,” IEEE Open Journal of the Communications Society, vol. 2, pp. 1015–1040, 2021.

[7] Amir Hossein Zarif, May 20, 2021, ”AoI Minimization in Energy Harvesting and Spectrum Sharing Enabled 6G Networks”, IEEE Dataport, doi: https://dx.doi.org/10.21227/aynt-my59.

[8] Adnan Fayyaz, Imran Mir, Faiza Gul, Suleman Mir, Nasir Saeed, Turke Althobaiti, Syed Manzar Abbas, and Laith Abualigah. 2022. ”Deep Reinforcement Learning for Integrated Non- Linear Control of Autonomous UAVs” Processes 10, no. 7: 1307. https://doi.org/10.3390/pr1007130

[9] Bander Alzahrani, Omar Sami Oubbati, Ahmed Barnawi, Mohammed Atiquzzaman, Daniyal Alghazzawi, UAV assistance paradigm: State-of-the-art in applications and challenges, Journal of Network and Computer Applications, Volume 166, 2020, ISSN 1084-8045, https://doi.org/10.1016/j.jnca.2020.102706 21

[10] Islam, N.; Rashid, M.M.; Pasandideh, F.; Ray, B.; Moore. S.; Kadel, R. A Review of Applications and Communication Technologies for Internet of Things (IoT) and Unmanned Aerial Vehicle (UAV) based Sustainable Smart Farming. Sustainability 2021, 13, 1821. Available at: https://doi.org/10.3390/su13041821

[11] F. Al-Turjman and H. Zahmatkesh, “A Comprehensive Review on the Use of AI in UAV Communications: Enabling Technologies, Applications, and Challenges,” Springer Unmanned Aerial Vehicles in Smart Cities, pp. 1–26, 2020.

[12] H. Zhou, C. She, Y. Deng, M. Dohler, and A. Nallanathan, “Machine learning for massive industrial internet of things,” IEEE Wireless Communications, vol. 28, no. 4, pp. 81–87, 2021.

[13] Z. Wei, M. Zhu, N. Zhang, L. Wang, Y. Zou, Z. Meng, H. Wu, and Z. Feng, “Uav-assisted data collection for internet of things: A survey,” IEEE Internet of Things Journal, vol. 9, no. 17, pp. 15 460–15 483, 2022.

[14] P. Asghari, A. M. Rahmani, and H. Haj Seyyed Javadi, “A medical monitoring scheme and health-medical service composition model in cloud-based iot platform,” Transactions on Emerging Telecommunications Technologies, vol. 30, no. 6, p. e3637, 2019.

[15] M. W. Woo, J. Lee, and K. Park, “A reliable iot system for personal healthcare devices,” Future Generation Computer Systems, vol. 78, pp. 626–640, 2018.

[16] A. Ahmad, S. Ahmad, M. H. Rehmani, and N. U. Hassan, “A survey on radio resource allocation in cognitive radio sensor networks,” IEEE Communications Surveys Tutorials, vol. 17, no. 2, pp. 888–917, 2015.

[17] J. Sanchez-Gomez, R. Sanchez-Iborra, and A. Skarmeta, “Transmission technologies comparison for iot communications in smart-cities,” in GLOBECOM 2017-2017 IEEE Global Communications Conference. IEEE, 2017, pp. 1–6.

[18] J.-S. Fu, Y. Liu, H.-C. Chao, B. K. Bhargava, and Z.-J. Zhang, “Secure data storage and searching for industrial iot by integrating fog computing and cloud computing,” vol. 14, no. 10, pp. 4519–4528, 2018.

[19] F. Liers, A. Martin, M. Merkert, N. Mertens, and D. Michaels, “Solving mixed-integer nonlinear optimization problems using simultaneous convexification: a case study for gas networks,” Journal of Global Optimization, vol. 80, no. 2, pp. 307–340, Feb. 2021. [Online]. Available: https://doi.org/10.1007/s10898-020-00974-0.

[20] W. Ahsan, W. Yi, Z. Qin, Y. Liu, and A. Nallanathan, “Resource allocation in uplink NOMA-IoT networks: A reinforcement-learning approach,” arXiv preprint arXiv:2007.08350, 2020.

[21] R. Zhang, X. Pang, J. Tang, Y. Chen, N. Zhao, and X. Wang, “Joint location and transmit power optimization for noma-uav networks via updating decoding order,” IEEE Wireless Communications Letters, vol. 10, no. 1, pp. 136–140, 2021.

[22] S. K. Singh, K. Agrawal, K. Singh, C.-P. Li, and Z. Ding, “Noma enhanced hybrid ris-uavassisted full-duplex communication system with imperfect sic and csi,” IEEE Transactions on Communications, vol. 70, no. 11, pp. 7609–7627, 2022.

[23] J. Fu, Y. Xiao, H. Liu, P. Yang, and B. Zhang, “A novel intelligent sic detector for noma systems based on deep learning,” in 2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring), 2021, pp. 1–6.

[24] Mahmud, S.K. (2022). Reinforcement Learning Empowered Unmanned Aerial Vehicle Assisted Internet of Things Networks. Doctoral thesis, Queen Mary University of London, School of Electronic Engineering and Computer Science, United Kingdom.

[25] L. Ardon, “Reinforcement learning to solve np-hard problems: an application to the CVRP,” CoRR, vol. abs/2201.05393, 2022. [Online]. Available: https://arxiv.org/abs/2201.05393

[26] S. Ray, “A quick review of machine learning algorithms,” in 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon), 2019, pp. 35–39.

[27] W. Ahsan, W. Yi, Z. Qin, Y. Liu, and A. Nallanathan, “Resource allocation in uplink NOMA-IoT networks: A reinforcement-learning approach,” arXiv preprint arXiv:2007.08350, 2020.

[28] M. Fayaz, W. Yi, Y. Liu, and A. Nallanathan, “Competitive ma-drl for transmit power pool design in semi-grant-free noma systems,” arXiv preprint arXiv:2106.11190, 2021.

[29] R. Zhong, X. Liu, Y. Liu, and Y. Chen, “Noma in uav-aided cellular offloading: A machine learning approach,” in 2020 IEEE Globecom Workshops (GC Wkshps, 2020, pp. 1–6.

[30] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT press, 2018.

[31] J. Cui, Y. Liu, and A. Nallanathan, “Multi-agent reinforcement learning-based resource allocation for uav networks,” vol. 19, no. 2, pp. 729–743, 2019.

[32] Khurram Mahmud, Syed; Chen, Yue; Chai, Kok Keong (2022): Tandem RL Framework for Sum Rate Enhancement in NOMA-UAV Network. TechRxiv. Preprint. https://doi.org/10.36227/techrxiv.20406972.v1

[33] J. Zhang, H. Zhu, F. Wang, J. Zhao, Q. Xu, and H. Li, “Security and privacy threats to federated learning: Issues, methods, and challenges,” Security and Communication Networks, vol. 2022, pp. 1–24, Sep. 2022. [Online]. Available: https://doi.org/10.1155/2022/2886795

[34] G. Xu, H. Li, H. Ren, K. Yang, and R. H. Deng, “Data security issues in deep learning: attacks, countermeasures, and opportunities,” IEEE Communications Magazine, vol. 57, no. 11, pp. 116–122, 2019.

[35] A. Alagil, M. Alotaibi, and Y. Liu, “Randomized positioning dsss for anti-jamming wireless communications,” in 2016 International Conference on Computing, Networking and Communications (ICNC), 2016, pp. 1–6.

[36] Available at: https://intellabs.github.io/coach/

[37] Understanding Activations and Optimization- Neural networks. (2023, February 18). Higher Logic, LLC. https://community.ibm.com/community/user/ai-datascience/blogs/pavan-saish naru/2023/01/27/optimization-hyperparameters

[38] Islam, N.; Rashid, M.M.; Pasandideh, F.; Ray, B.; Moore. S.; Kadel, R. A Review of Applications and Communication Technologies for Internet of Things (IoT) and Unmanned Aerial Vehicle (UAV) based Sustainable Smart Farming. Sustainability 2021, 13, 1821. Available at: https://doi.org/10.3390/su13041821

[39] Analytical Review on OMA vs. NOMA and Challenges Implementing NOMA, Second International Conference on Smart Electronics and Communication (ICOSEC), 2021. Available at: IEEE Xplore Part Number: CFP21V90-ART; ISBN: 978-1-6654-3368-6

[40] Adnan Fayyaz, Imran Mir, Faiza Gul, Suleman Mir, Nasir Saeed, Turke Althobaiti, Syed Manzar Abbas, and Laith Abualigah. 2022. ”Deep Reinforcement Learning for Integrated Non- Linear Control of Autonomous UAVs” Processes 10, no. 7: 1307. https://doi.org/10.3390/pr1007130