UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA
DELL'INFORMAZIONE

CORSO DI LAUREA IN ICT FOR INTERNET AND
MULTIMEDIA

REINFORCEMENT LEARNING BASED ALGORITHM FOR ANTENNA SELECTION

Relatore:
Prof. Stefano Tomasin

Laureando:
Federico Danesin

ANNO ACCADEMICO: 2024/2025
Data di laurea: 27/03/2025

Alla mia Famiglia, ai miei Amici e a tutte quelle persone che hanno reso la mia vita una fantastica storia.

Contents

Abstract

This thesis explores reinforcement learning approaches for the antenna selection problem in Wi-Fi devices, aiming to provide adaptive solutions to the dynamic nature of wireless communication channels. The problem is mathematically modeled using Markov Decision Processes (MDPs), and a reinforcement learning framework is implemented to address this challenge. The study evaluates the performance of a Proximal Policy Optimization (PPO) algorithm, comparing it with traditional approaches. A key objective of this research is to apply reinforcement learning and machine learning to real-world problems, demonstrating their advantages in optimizing antenna selection in dynamic communication scenarios.

List of Acronyms

Table 1: List of Acronyms

Acronym	**Definition**
AWGN	Additive White Gaussian Noise
CSI	Channel State Information
DFT	Discrete Fourier Transform
FFT	Fast Fourier Transform
IDFT	Inverse Discrete Fourier Transform
ISI	Inter-Symbol Interference
LTI	Linear Time-Invariant
MIMO	Multiple-Input Multiple-Output
OFDM	Orthogonal Frequency Division Multiplexing
OFDMA	Orthogonal Frequency Division Multiple Access
QoS	Quality of Service
RSSI	Received Signal Strength Indicator
RU	Resource Unit
SFO	Sampling Frequency Offset

Acronym	Definition
STO	Sampling Time Offset
WLAN	Wireless Local Area Network
RL	Reinforcement Learning
MDP	Markov Decision Process
POMDP	Partially Observable Markov Decision Process
DRL	Deep Reinforcement Learning
DNN	Deep Neural Network
DQN	Deep Q-Networks
Q-Learning	Q-Learning Algorithm
TD	Temporal Difference
A2C	Advantage Actor-Critic
GAE	Generalized Advantage Estimation
PPO	Proximal Policy Optimization
PER	Prioritized Experience Replay
QAM	Quadrature Amplitude Modulation
PSK	Phase Shift Keying

Acronym	Definition
CP	Cyclic Prefix
AP	Access Point
MAC	Medium Access Control
PHY	Physical Layer
BSS	Basic Service Set
STC	Space-Time Coding
SNR	Signal-to-Noise Ratio
MU-MIMO	Multi-User Multiple-Input Multiple-Output
EM	Electromagnetic
RF	Radio Frequency
DSSS	Direct-Sequence Spread Spectrum
HE-SIG-B	High-Efficiency Signal-B
RNN	Recurrent Neural Network
GRU	Gated Recurrent Unit
SVD	Singular Value Decomposition

Introduction

The rapid expansion of wireless communication systems has driven an increasing demand for higher data rates, improved reliability, and enhanced spectral efficiency. Modern wireless networks must efficiently manage limited spectrum resources while adapting to dynamic and often unpredictable channel conditions. One of the critical aspects in achieving these objectives is antenna selection, a technique that dynamically optimizes signal reception and transmission by choosing the most suitable subset of available antennas at any given time.

With the advent of IEEE 802.11ax, commonly referred to as Wi-Fi 6, the need for adaptive and intelligent mechanisms to optimize network performance has become even more pronounced. Traditional antenna selection strategies often rely on predefined heuristics or static configurations, which, while effective in specific scenarios, tend to lack the adaptability required to cope with the ever-changing nature of wireless environments. Consequently, there has been a growing interest in leveraging artificial intelligence (AI) and machine learning techniques to enhance the efficiency of wireless communication systems. Among these approaches, reinforcement learning (RL) has emerged as a particularly promising solution due to its ability to learn optimal

decision-making strategies through direct interaction with the environment.

This thesis explores the application of reinforcement learning to the antenna selection problem in Wi-Fi networks, aiming to improve network performance by dynamically optimizing antenna configurations in response to changing channel conditions. By formulating antenna selection as a Markov Decision Process (MDP), this work defines a framework where an intelligent agent can learn to make optimal decisions based on real-time network conditions. The deep reinforcement learning (DRL) approach employed in this study is based on Proximal Policy Optimization (PPO), a policy gradient algorithm that balances exploration and exploitation while ensuring stable and efficient learning. PPO is particularly well-suited to this task, as it mitigates the instability and computational challenges associated with traditional policy gradient methods, making it a robust choice for real-world wireless communication applications.

The methodology presented in this thesis involves the development of an RL agent trained to select antenna configurations based on key performance indicators (KPIs) such as channel frequency response, received signal strength, and application-level throughput. The performance of the proposed approach is evaluated in a real-world testing environment, comparing its effectiveness against a baseline heuristic policy that relies on static selection strategies. The experimental results demonstrate that the RL-based method significantly improves throughput and adaptability, particularly in dynamic channel conditions where traditional approaches struggle to maintain optimal performance.

Beyond presenting a novel approach to antenna selection, this work con-

tributes to the broader field of intelligent wireless network optimization. By demonstrating the feasibility of applying RL techniques in practical Wi-Fi deployments, it paves the way for future research on AI-driven network management. The insights gained from this study not only highlight the advantages of reinforcement learning in wireless communication but also open new possibilities for integrating AI-based optimization in emerging technologies such as Wi-Fi 7 and beyond.

This thesis is structured to provide a comprehensive exploration of the fundamental concepts, methodological framework, and experimental findings. The first chapter introduces the key technologies underpinning high-efficiency wireless networks, including Wi-Fi 6, MIMO, OFDM, and channel modeling, providing the necessary background for understanding the antenna selection problem. The second chapter delves into the theoretical foundations of reinforcement learning, explaining the principles of Markov Decision Processes, policy optimization, and deep reinforcement learning models, culminating in the discussion of PPO, the algorithm chosen for this study. The third chapter formalizes the problem of antenna selection in the RL framework, detailing the system model, the design of the state space, action space, and reward function, as well as the implementation of the PPO-based learning agent. This chapter also presents the performance evaluation, analyzing the results obtained from real-world experiments and comparing them with the baseline policy. Finally, the conclusion summarizes the key findings, discusses the practical implications of this work, and outlines potential future research directions.

The increasing complexity of wireless networks necessitates advanced op-

timization techniques capable of adapting to diverse and evolving communication environments. This research demonstrates that reinforcement learning provides a powerful framework for optimizing antenna selection in Wi-Fi networks, enabling improved spectral efficiency, enhanced network adaptability, and reduced reliance on predefined heuristics. By bridging the gap between AI-driven decision-making and practical wireless communication applications, this study lays the foundation for the next generation of intelligent, self-optimizing networks.

Chapter 1

Building Blocks of High-Efficiency Wireless Networks

In this chapter, we aim to provide a concise yet comprehensive introduction to Wi-Fi standards, with a particular emphasis on the technological advancements that have underpinned recent developments in wireless communication. The Wi-Fi standard, formally based on IEEE 802.11 specifications, has evolved to accommodate increasing demands for data rates, efficiency, and device density. This overview will highlight the key features and functionality of Wi-Fi standards, especially those that are crucial to understanding advanced Wi-Fi technologies and their applications in modern networks.

Key topics addressed in this chapter include Orthogonal Frequency-Division Multiplexing (OFDM) and Multiple-Input Multiple-Output (MIMO) systems, both of which are foundational to the increased throughput and ro-

bustness of current Wi-Fi standards. OFDM modulation is a significant technological advancement that allows for efficient use of bandwidth and reduced interference by dividing a high-rate data stream into multiple lower-rate streams that are transmitted simultaneously over various orthogonal subcarriers. This technique not only improves spectral efficiency but also enhances the resilience of the system to multipath fading.

MIMO technology further enhances Wi-Fi performance by leveraging multiple antennas at both the transmitter and receiver to improve data transmission rates and reliability. Through spatial diversity and spatial multiplexing, MIMO systems can achieve higher throughput and better signal quality, making them essential for high-capacity, low-latency applications in dense network environments.

In addition, this chapter delves into channel estimation methods and the concept of Channel State Information (CSI). Accurate channel estimation is critical for optimizing signal processing and adapting transmission parameters to real-time conditions in the wireless environment. CSI provides detailed information about the state of the wireless channel, enabling more effective resource allocation, adaptive modulation, and power control, all of which are fundamental to achieving high data rates and efficient spectrum usage in modern Wi-Fi networks.

These technological concepts form the foundation for advanced research and development in wireless communication and will be pivotal in addressing future challenges in Wi-Fi, such as enhancing Quality of Service (QoS) in diverse and demanding use cases. The subsequent sections will expand on each of these topics, offering insights into the technical mechanisms and

practical implications that define the current state of Wi-Fi technology and its potential for future innovation.

1.1 Wi-Fi introduction

Wi-Fi, or Wireless Fidelity, refers to a set of wireless network protocols that allow devices to communicate and transfer data over radio waves, without physical cables. This technology operates within the electromagnetic (EM) spectrum, primarily in the radio frequency (RF) bands ranging from 2.4 GHz to 5 GHz, and, in recent advancements, extends to the 6 GHz band. Radio waves used in Wi-Fi, being a form of EM radiation, possess longer wavelengths than visible light, making them suitable for wireless data transmission across various environments.

The Wi-Fi protocols are based on the IEEE 802.11 family of standards, which specify how devices communicate over wireless local area networks (WLANs). The IEEE 802 standards encompass various technologies for wired and wireless networking, but IEEE 802.11 specifically addresses wireless communication through protocols that define both the Medium Access Control (MAC) and Physical Layer (PHY) aspects. Devices certified under Wi-Fi standards are capable of connecting with each other reliably, regardless of the manufacturer, due to interoperability ensured by the Wi-Fi Alliance.

Wi-Fi has gone through several iterations, each marked by improvements in speed, range, and efficiency. Early standards like IEEE 802.11b utilized Direct-Sequence Spread Spectrum (DSSS) modulation, enabling basic wireless communication. Later versions, such as 802.11a and 802.11n, adopted

OFDM, which increased efficiency and speed. The most recent standards, Wi-Fi 5 (802.11ac) and Wi-Fi 6 (802.11ax), provide significantly higher data rates and are optimized for dense environments, with Wi-Fi 6 adding OFDMA to manage multiple users simultaneously with minimal interference.

Wi-Fi operates over several frequency bands, primarily 2.4 GHz and 5 GHz, with the addition of 6 GHz in the Wi-Fi 6E standard. The 2.4 GHz band offers broader coverage but is susceptible to interference from other devices, such as microwaves and Bluetooth devices. The 5 GHz band, while having a shorter range, provides additional channels and supports higher data rates due to less interference. Within these frequency bands, Wi-Fi channels are defined at specific intervals, generally 5 MHz apart. To avoid interference, Wi-Fi devices require channel separations to ensure clear communication; for example, the 2.4 GHz band allows only three non-overlapping channels due to its narrower range, while the 5 GHz band, with its wider frequency range, supports up to 23 non-overlapping channels in certain regions.

As the demand for faster and more efficient Wi-Fi grows, standards have evolved to allow for wider channel bandwidths. For instance, 802.11n introduced 40 MHz channels by doubling the channel width in the 2.4 GHz band, while 802.11ac and 802.11ax provide options for 80 MHz and 160 MHz channels in the 5 GHz and 6 GHz bands, enhancing data throughput for high-demand applications. This flexibility in channel width and the ability to aggregate multiple channels have become crucial in meeting the data needs of modern devices and applications. The IEEE 802.11 working group is responsible for developing specifications for Wireless Local Area Networks (WLAN). Since its inception in the late 1990s, the group has introduced

several successful standards, including 802.11a, 802.11b, and 802.11g. Today, WLAN is ubiquitous, integrated as a standard feature in most laptops, smartphones, and other devices. Since this thesis work involves WiFi 6 devices, we will explore WiFi 6 specifications in greater detail in the following sections.

1.2 Wireless Channel Models

1.2.1 Introduction to the Wireless Channel

The wireless channel plays a crucial role in determining the reliability and efficiency of wireless communication systems. It is characterized by variations in signal strength over time, space, and frequency due to phenomena such as reflection, diffraction, scattering, and multipath propagation. These variations can be broadly categorized into:

- **Large-scale fading:** Caused by path loss and shadowing due to large obstacles (e.g., buildings, hills), occurring over distances comparable to the cell size.

- **Small-scale fading:** Caused by the constructive and destructive interference of multipath components, occurring at scales comparable to the wavelength of the signal.

Understanding and modeling these phenomena are essential for optimizing system design, including power control, modulation, and detection.

1.2.2 Physical Phenomena in Wireless Channels

Wireless communication relies on electromagnetic (EM) waves. However, the interaction of EM waves with the environment introduces several propagation effects:

Path Loss

Path loss is the attenuation of signal power as it propagates through space. In free space, the received power P_r decreases with the square of the distance r, given by the Friis transmission equation:

$$P_r = P_t G_t G_r \left(\frac{\lambda}{4\pi r}\right)^2 \tag{1.1}$$

where:

- P_t and P_r are the transmitted and received power, respectively.
- G_t and G_r are the transmitter and receiver antenna gains.
- λ is the wavelength.

Shadowing

Shadowing occurs due to obstacles (e.g., buildings, vehicles) between the transmitter and receiver. These obstacles attenuate the signal power through absorption, reflection, and scattering.

- **Absorption:** Electromagnetic energy is absorbed by matter, converting it into heat. This leads to power attenuation.

- **Reflection:** Large surfaces, such as walls, reflect the signal, introducing constructive or destructive interference.
- **Diffraction:** Waves bend around obstacles with sharp edges or small openings, allowing signal propagation even without a direct line of sight.
- **Scattering:** Small objects cause the incident wave to radiate in multiple directions, altering the signals path.

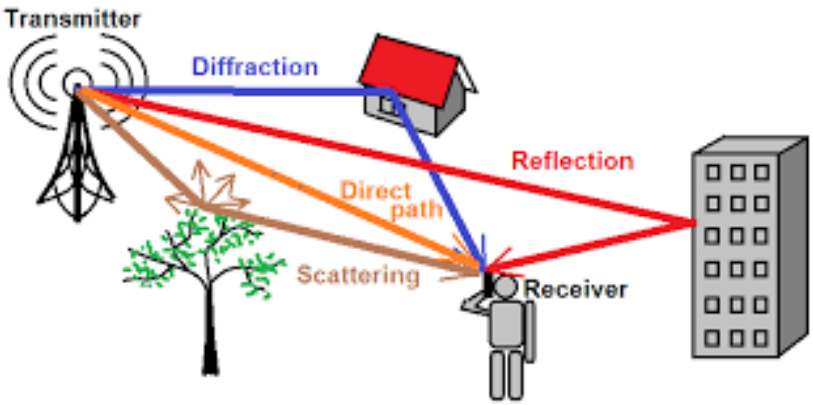


Figure 1.1: Radio propagation phenomena

Multipath Propagation

Multipath propagation occurs when multiple signal paths exist between the transmitter and receiver due to reflections, diffraction, or scattering. The received signal can be expressed as:

$$y(t) = \sum_{i=1}^{N} \rho_i e^{j\phi_i} x(t - \tau_i) \tag{1.2}$$

where:

- N is the number of paths,
- ρ_i and ϕ_i are the amplitude and phase of the i-th path,
- τ_i is the delay of the i-th path.

Multipath causes **delay spread**, defined as the difference in propagation times between the shortest and longest paths:

$$T_d = \tau_{\max} - \tau_{\min} \tag{1.3}$$

1.2.3 Key Channel Parameters

The behavior of a wireless channel is often described using the following parameters:

Coherence Bandwidth

The coherence bandwidth W_c is the frequency range over which the channel response remains approximately constant. It is inversely proportional to the delay spread:

$$W_c = \frac{1}{2T_d} \tag{1.4}$$

Doppler Spread and Coherence Time

When either the transmitter or receiver is moving, the frequency of the received signal shifts due to the Doppler effect. The Doppler spread D_s quantifies the range of frequency shifts, while the coherence time T_c represents

the time duration over which the channel remains stationary:

$$T_c = \frac{1}{4D_s} \tag{1.5}$$

Flat vs. Frequency-Selective Fading

Channels are categorized based on their bandwidth relative to the coherence bandwidth:

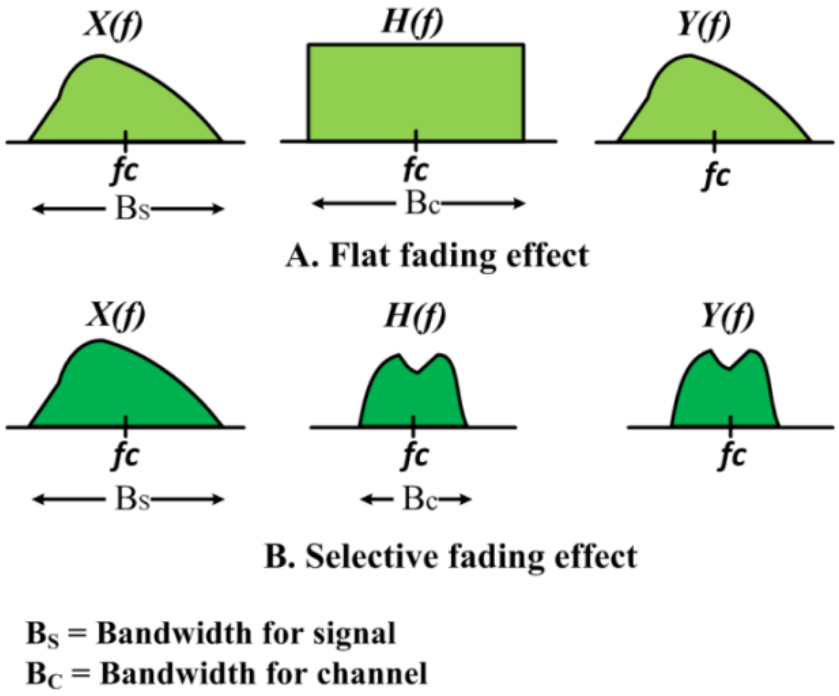


Figure 1.2: The effect of flat and selective fading channel on the signal

- **Flat fading:** Occurs when $W \ll W_c$. The channel can be modeled as a single tap.
- **Frequency-selective fading:** Occurs when $W \gg W_c$. Multiple taps are needed to model the channel.

1.3 Input/Output Model of the Wireless Channel

This section presents an input/output model of the wireless channel, describing multipath propagation effects as a linear time-varying system. A baseband representation is then introduced, followed by the sampling of the continuous-time channel to obtain a discrete-time model, concluding with the inclusion of additive noise.

1.3.1 Linear Time-Varying Model

The multipath channel can be modeled as a linear time-varying system:

$$y(t) = \sum_i a_i(t)x(t - \tau_i(t)) \tag{1.6}$$

where:

- $a_i(t)$: overall attenuation on the i-th path,
- $\tau_i(t)$: propagation delay on the i-th path,
- $x(t)$: transmitted signal,
- $y(t)$: received signal.

The impulse response of the channel is given by:

$$h(t, \tau) = \sum_i a_i(t)\delta(\tau - \tau_i(t)). \tag{1.7}$$

The input/output relation of the system can be expressed as:

$$y(t) = \int h(t,\tau)x(t-\tau)\,d\tau. \tag{1.8}$$

In stationary conditions (time-invariant channel):

$$h(\tau) = \sum_i a_i \delta(\tau - \tau_i). \tag{1.9}$$

1.3.2 Passband and Baseband Relationships

Wireless communication systems process signals at the baseband to simplify operations. The passband signal $x(t)$ is obtained from the baseband signal $x_b(t)$ as:

$$x(t) = \sqrt{2}\mathrm{Re}\left\{x_b(t)e^{j2\pi f_c t}\right\}. \tag{1.10}$$

The corresponding Fourier transform relationship is:

$$X(f) = \frac{1}{\sqrt{2}}\left[X_b(f-f_c) + X_b^*(-f-f_c)\right], \tag{1.11}$$

where $X(f)$ and $X_b(f)$ are the spectra of $x(t)$ and $x_b(t)$, respectively.

The baseband signal can be reconstructed from the passband signal using:

$$x_b(t) = \sqrt{2}\int_{-\infty}^{\infty} X(f+f_c)e^{j2\pi f t}df. \tag{1.12}$$

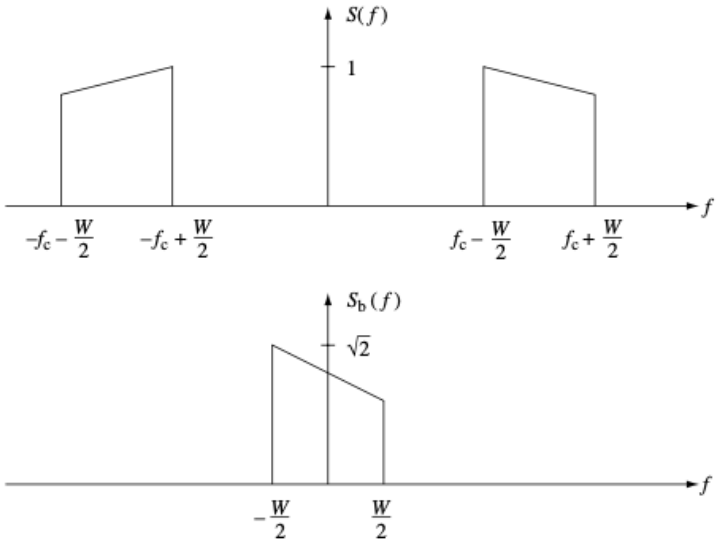


Figure 1.3: Relationship between passband and baseband spectra.

1.3.3 Baseband Equivalent Model

The baseband equivalent of the multipath fading channel is:

$$y_b(t) = \sum_i a_i(t) e^{-j2\pi f_c \tau_i(t)} x_b(t - \tau_i(t)). \tag{1.13}$$

The baseband impulse response is:

$$h_b(t, \tau) = \sum_i a_i(t) e^{-j2\pi f_c \tau_i(t)} \delta(\tau - \tau_i(t)). \tag{1.14}$$

1.3.4 Discrete-Time Model

Sampling the baseband signal at intervals of $T_s = 1/W$ leads to the discrete-time model:

$$y[m] = \sum_{\ell} h_\ell[m] x[m - \ell], \tag{1.15}$$

where:

- $h_\ell[m]$: the ℓ-th filter tap at time m,
- $x[m]$ and $y[m]$: discrete-time transmitted and received signals.

The filter tap coefficients $h_\ell[m]$ are defined as:

$$h_\ell[m] = \sum_{i} a_i[m] \text{sinc}(W(\ell - \tau_i[m]W)), \tag{1.16}$$

where W is the bandwidth.

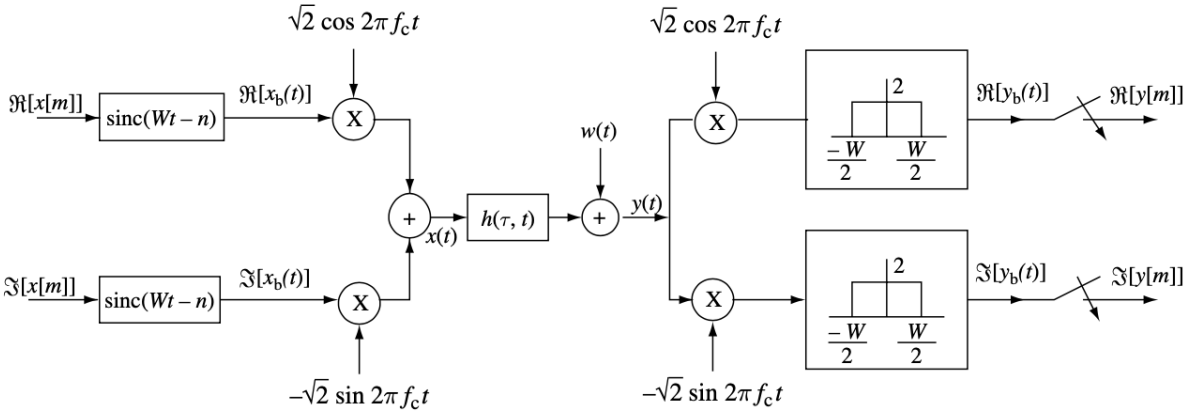


Figure 1.4: Discrete-time system diagram.

1.3.5 Inclusion of Additive Noise

To model noise effects, an additive white Gaussian noise (AWGN) term is included:

$$y[m] = \sum_{\ell} h_{\ell}[m]x[m-\ell] + w[m], \tag{1.17}$$

where $w[m]$ is a stationary Gaussian process with zero mean and variance $N_0/2$.

1.3.6 Summary of Parameters

Table 1.1: Key parameters of the wireless channel model.

Parameter	**Symbol**	**Typical Value**
Carrier frequency	f_c	1 GHz
Bandwidth	W	1 MHz
Coherence time	T_c	2.5 ms
Doppler spread	D_s	100 Hz
Multipath delay spread	T_d	1 μs
Coherence bandwidth	W_c	500 kHz

1.4 OFDM

OFDM is an advanced transmission technique that transforms communication over frequency-selective channels into communication over simpler parallel narrowband subchannels. The theoretical basis of OFDM relies on the property that sinusoids are eigenfunctions of linear time-invariant (LTI) systems, making the frequency domain particularly useful for handling frequency-selective channels.

Concept of Orthogonality in OFDM

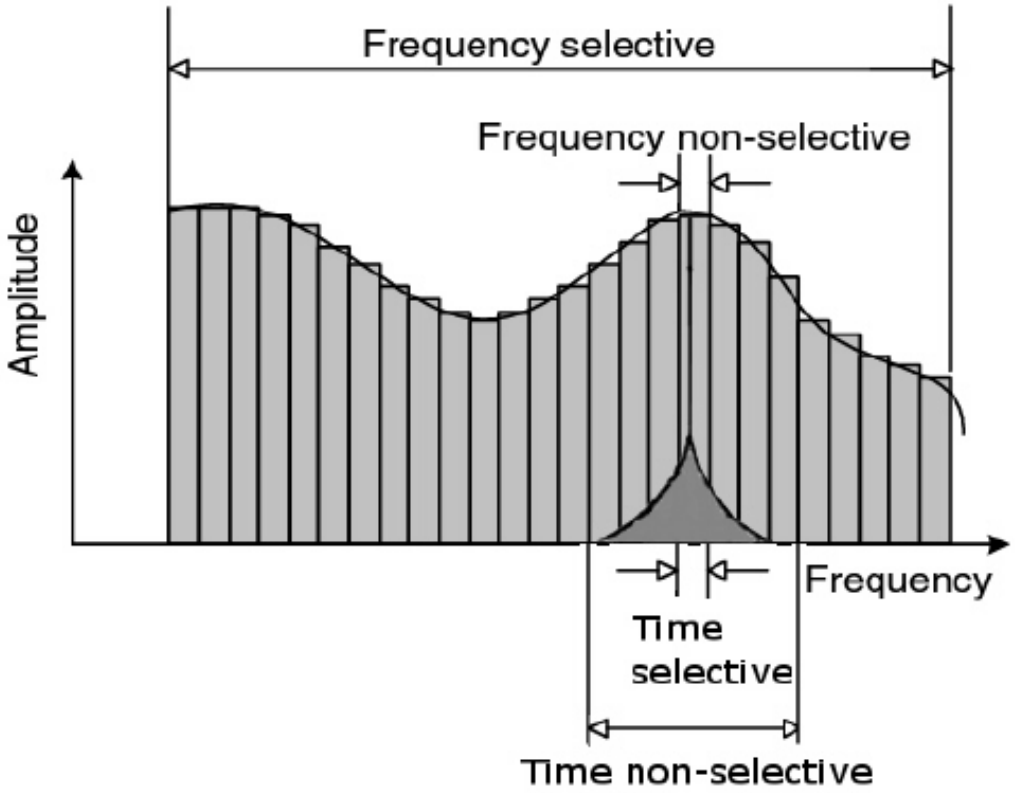


Figure 1.5: OFDM Channel Division

In OFDM, the available bandwidth is divided into several closely spaced, orthogonal subcarriers, each carrying a portion of the data stream. Orthogonality between subcarriers ensures that the peak of each subcarrier aligns with the zero points of others, effectively reducing inter-carrier interference (ICI). This orthogonality is achieved by carefully spacing subcarrier frequencies. The frequency spacing Δf is defined as:

$$\Delta f = \frac{1}{T_s}, \tag{1.18}$$

where T_s is the symbol duration. With this spacing, each subcarrier completes an integer number of cycles over T_s, ensuring that the integral of the product of any two subcarrier signals over T_s is zero, maintaining orthogonality.

Cyclic Prefix

To preserve the orthogonality between OFDM subchannels and mitigate inter-symbol interference (ISI), a cyclic prefix (CP) is added. For a block of symbols $d = [d_0, d_1, \ldots, d_{N_c-1}]^T$, the extended input block becomes:

$$x = [d_{N_c-L+1}, \ldots, d_{N_c-1}, d_0, \ldots, d_{N_c-1}]^T. \tag{1.19}$$

As shown in Figure 1.6, the cyclic prefix consists of the tail symbols of the original block replicated at the beginning.

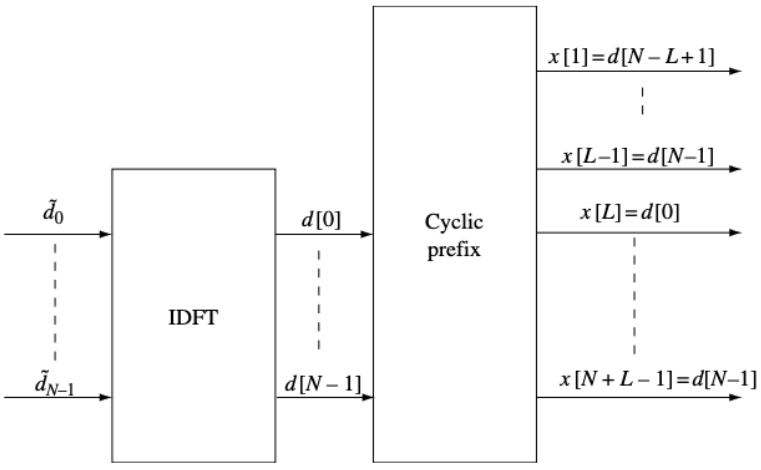


Figure 1.6: Addition of the cyclic prefix to the symbol block.

At the receiver, only the useful symbols are considered after discarding the cyclic prefix. The received output is given by:

$$y_m = \sum_{\ell=0}^{L-1} h_\ell d_{(m-\ell) \mod N_c} + w_m, \tag{1.20}$$

where the modulo N_c ensures the periodicity imposed by the cyclic prefix.

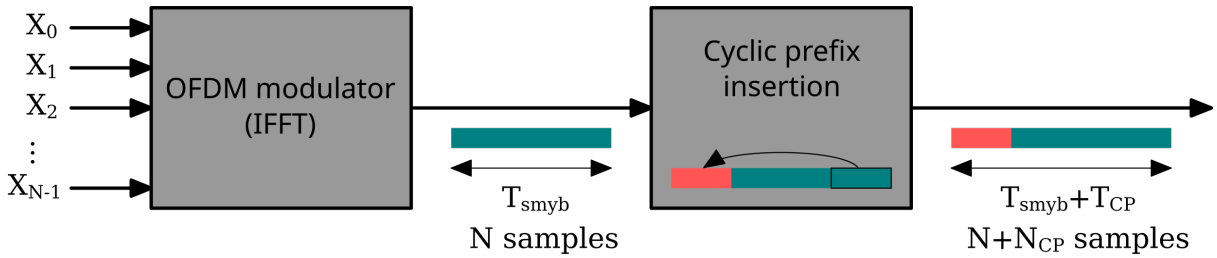


Figure 1.7: OFDM Symbol with Cyclic Prefix

Discrete Fourier Transform (DFT) Representation

Applying the DFT to the system yields:

$$\tilde{y}_n = \tilde{h}_n \tilde{d}_n + \tilde{w}_n, \quad n = 0, \ldots, N_c - 1, \tag{1.21}$$

where:

$$\tilde{h}_n = \sum_{\ell=0}^{L-1} h_\ell e^{-j\frac{2\pi n\ell}{N_c}} \tag{1.22}$$

represents the channel's frequency response. The DFT diagonalizes the circular convolution, resulting in orthogonal subchannels.

The CP introduces an overhead in both time and power. The time over-

head is the fraction of time occupied by the CP:

$$\text{Time Overhead} = \frac{L}{N+L}. \tag{1.23}$$

Similarly, the power overhead arises because the cyclic prefix does not carry useful data. To minimize these overheads, the block length N should be chosen to satisfy:

$$T_c W \gg N \gg L, \tag{1.24}$$

where T_c is the channel coherence time.

1.4.1 OFDM Transmission Scheme

The transmission and reception scheme for an OFDM system is illustrated in Figure 1.8. In this scheme, data symbols are modulated onto N orthogonal subcarriers, converted to the time domain using the Inverse Discrete Fourier Transform (IDFT), and transmitted along with a cyclic prefix. At the receiver, the cyclic prefix is removed, and the DFT is applied to recover the data symbols.

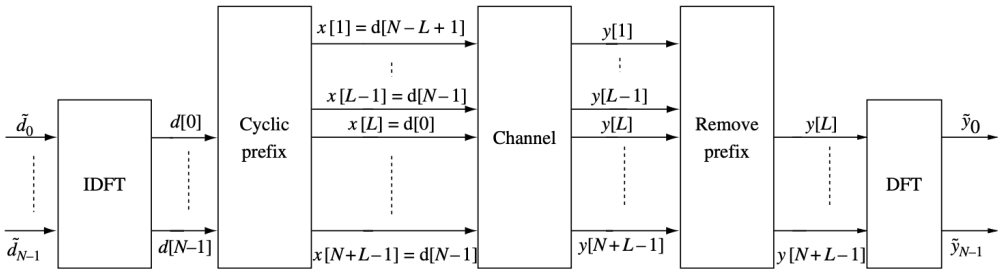


Figure 1.8: OFDM transmission and reception scheme.

Transmitter

The input is a serial stream of binary data, $s[n]$, which is divided into N parallel streams via a serial-to-parallel conversion. Each stream is mapped to a modulation constellation such as Quadrature Amplitude Modulation (QAM) or Phase Shift Keying (PSK), producing frequency-domain symbols X_k, where $k = 0, 1, \ldots, N-1$.

The IDFT is then applied to convert the frequency-domain symbols into time-domain samples:

$$x[n] = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} X_k e^{j2\pi \frac{kn}{N}}, \quad n = 0, 1, \ldots, N-1, \tag{1.25}$$

where:

- X_k: Modulated data symbols,
- N: Number of subcarriers,
- $x[n]$: Time-domain OFDM samples.

To mitigate inter-symbol interference (ISI), a cyclic prefix of length L is prepended to each block, creating the transmitted signal:

$$x_{\text{cp}}[n] = x[(n \mod N)], \quad n = -L, \ldots, N-1. \tag{1.26}$$

Finally, the signal is converted to analog using Digital-to-Analog Converters (DACs) and modulated to a carrier frequency f_c for transmission.

Receiver

The receiver captures the signal $r(t)$, downconverts it to baseband, and digitizes it using Analog-to-Digital Converters (ADCs). After removing the cyclic prefix, the remaining samples are passed through a DFT to transform them back into the frequency domain:

$$Y[k] = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} r[n] e^{-j2\pi \frac{kn}{N}}, \quad k = 0, 1, \ldots, N-1. \tag{1.27}$$

The output $Y[k]$ corresponds to the received data symbols affected by the channel response $H[k]$ and noise $W[k]$:

$$Y[k] = H[k]X_k + W[k]. \tag{1.28}$$

Here:

- $H[k]$: Frequency response of the channel,
- $W[k]$: Additive noise in the frequency domain.

The received symbols are then demodulated using a symbol detector to estimate the original data stream $\hat{s}[n]$.

1.4.2 Frequency Diversity

Frequency diversity is achieved in OFDM by spreading data across multiple uncorrelated subchannels. The coherence bandwidth W_c, inversely proportional to the channel delay spread T_d, determines the number of independent

subchannels:

$$W_c = \frac{1}{2T_d}. \tag{1.29}$$

To maximize diversity, the subcarrier spacing $\Delta f = \frac{W}{N}$ must be much smaller than W_c. This ensures that data on adjacent subcarriers experiences independent fading, allowing for efficient error correction and robust communication.

The DFT and IDFT operations in OFDM can be efficiently implemented using the Fast Fourier Transform (FFT) and its inverse (IFFT). This significantly reduces computational complexity, particularly when N is a power of 2.

1.5 OFDMA

OFDMA is an extension of OFDM, designed for environments where multiple users require concurrent access to the same channel. OFDMA divides subcarriers into Resource Units (RUs), allocating them to different users based on bandwidth needs, thereby reducing latency and enhancing spectral efficiency.

1.5.1 Resource Units in OFDMA

The bandwidth in an OFDMA system is split into RUs, each composed of multiple subcarriers. RUs allow Wi-Fi 6 access points to assign channel resources dynamically:

$$RU_i = \{f_{i,1}, f_{i,2}, \ldots, f_{i,k}\}, \tag{1.30}$$

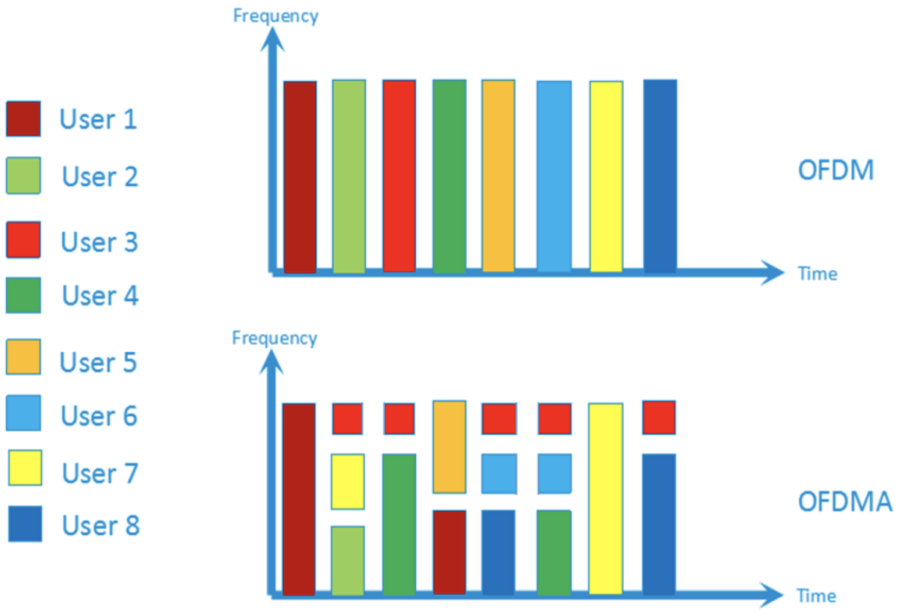


Figure 1.9: Comparison of OFDM and OFDMA

where $f_{i,k}$ represents the k-th subcarrier within the i-th RU. The smallest RU is defined with 26 subcarriers, and larger RUs can have 52, 106, or more subcarriers, depending on the channel bandwidth.

1.5.2 OFDMA in Downlink and Uplink

In the downlink direction, the access point (AP) uses OFDMA to transmit data to multiple users simultaneously. The AP manages RU assignments, embedding user-specific information in the High-Efficiency Signal-B (HE-SIG-B) field. In the uplink, the AP coordinates users with a trigger frame that specifies transmission parameters such as power and timing, ensuring that all users maintain synchronization.

RU Size	20 MHz	40 MHz	80 MHz	160 MHz
26 subcarriers	9	18	37	74
52 subcarriers	4	8	16	32
106 subcarriers	2	4	8	16
242 subcarriers	1	2	4	8
484 subcarriers	N/A	1	2	4
996 subcarriers	N/A	N/A	1	2

Table 1.2: Resource Units Allocation in Different Channel Widths

$$s_u(t) = \sum_{k=1}^{N_u} x_{u,k} e^{j2\pi f_{u,k} t}, \tag{1.31}$$

where N_u is the number of subcarriers assigned to user u, $x_{u,k}$ is the data symbol for the k-th subcarrier, and $f_{u,k}$ is the frequency of the k-th subcarrier for user u. OFDMA enables efficient multi-user access by dynamically assigning RUs, reducing latency and minimizing contention in dense environments. It is also highly flexible, allowing simultaneous support for users with varying bandwidth requirements. The OFDMA scheme is sensitive to synchronization issues, particularly frequency offset and Doppler shift, which can compromise orthogonality among subcarriers. To mitigate these issues, precise time and frequency synchronization mechanisms are essential. OFDM and its multi-access extension OFDMA are pivotal technologies in Wi-Fi 6, supporting high efficiency, robust spectral utilization, and enhanced multi-user capabilities. Their deployment in modern communication systems marks a significant advancement in achieving reliable, high-speed, and low-latency wireless connectivity.

1.6 MIMO Systems

1.6.1 Introduction to MIMO

MIMO is a cutting-edge technology designed to enhance the capacity and reliability of wireless communication systems. By utilizing multiple antennas at both the transmitter and receiver, MIMO exploits the multipath propagation of radio signals, effectively transforming what was once a limitation into an advantage.

Initially, MIMO referred to the simple use of multiple antennas at both ends of the communication link. Modern MIMO, however, encompasses a set of advanced techniques for simultaneously sending and receiving multiple data streams over the same radio channel. This is achieved by leveraging differences in spatial signatures of the signals induced by multipath propagation. The use of OFDM further complements MIMO by providing robustness against multipath interference and enhancing data rates.

1.6.2 MIMO Techniques

MIMO techniques are typically categorized into the following three main strategies:

Precoding

Precoding is a transmission strategy that maximizes the signal power at the receiver. It involves sending the same signal from each transmit antenna with appropriate phase and gain weighting. This process requires knowledge

of Channel State Information (CSI) at both the transmitter and receiver. Precoding ensures that signals from different antennas add constructively at the receiver, increasing the effective signal-to-noise ratio (SNR).

Spatial Multiplexing

Spatial Multiplexing (SM) divides a high-rate data stream into multiple lower-rate streams, transmitting each stream from a different antenna using the same frequency channel. The receiver must have accurate CSI to separate the streams into parallel channels. SM significantly increases channel capacity, particularly in environments with a high SNR and rich multipath.

Diversity Coding

Diversity coding is employed when CSI is unavailable at the transmitter. It transmits a single data stream using space-time coding (STC), which encodes redundant copies of the data across multiple antennas. This technique exploits the independent fading of multipath channels to improve signal robustness and reliability.

1.6.3 MIMO Channel Modeling and Capacity Analysis

To introduce the fundamentals of MIMO systems, we consider a narrowband point-to-point MIMO channel. A communication setup with M_T transmitting antennas and M_R receiving antennas can be described by the following discrete-time model:

$$\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{n} \tag{1.32}$$

where:

- $\mathbf{x}$ is the M_T-dimensional transmitted signal vector,
- $\mathbf{y}$ is the M_R-dimensional received signal vector,
- $\mathbf{H}$ is the $M_R \times M_T$ channel matrix whose entries h_{ij} represent the channel gain from the j-th transmit antenna to the i-th receive antenna,
- $\mathbf{n}$ is the M_R-dimensional noise vector modeled as complex Gaussian noise with zero mean and covariance $\sigma_n^2 \mathbf{I}_{M_R}$.

The channel capacity in a MIMO system is defined as the maximum mutual information $I(\mathbf{s}; \mathbf{y})$ between the transmitted signal $\mathbf{s}$ and the received signal $\mathbf{y}$:

$$C = \max_{f(\mathbf{s})} I(\mathbf{s}; \mathbf{y}) \tag{1.33}$$

Assuming no Channel State Information (CSI) at the transmitter and that the input signals have an identity covariance matrix ($\mathbf{R}_{SS} = \mathbf{I}_{M_T}$), the channel capacity simplifies to:

$$C = \log_2 \det \left(\mathbf{I}_{M_R} + \frac{E_S}{M_T N_0} \mathbf{H}\mathbf{H}^H \right) \tag{1.34}$$

where E_S denotes the total transmitted power, and N_0 is the noise power spectral density.

When the channel matrix $\mathbf{H}$ is perfectly known at the transmitter, it is possible to apply singular value decomposition (SVD) to decompose the channel into independent sub-channels:

$$\mathbf{H} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H \tag{1.35}$$

where $\mathbf{\Sigma}$ is a diagonal matrix containing the singular values $\sqrt{\lambda_i}$. With optimal power distribution using the water-filling algorithm, the capacity becomes:

$$C = \sum_{i=1}^{r} \log_2 \left(1 + \frac{\gamma_i \lambda_i}{N_0}\right) \tag{1.36}$$

where $\gamma_i^{\text{opt}} = \max\left(\mu - \frac{N_0}{\lambda_i}, 0\right)$ represents the optimal power allocation for each sub-channel.

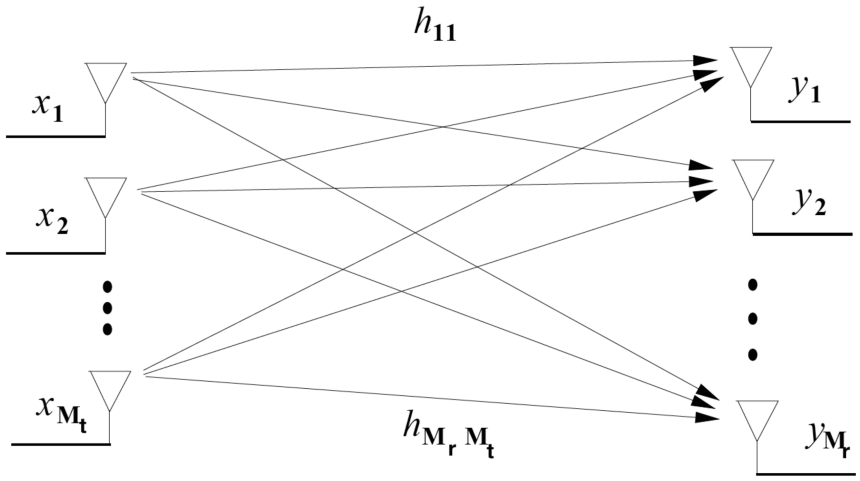


Figure 1.10: Illustration of a MIMO system with M_T transmitting and M_R receiving antennas.

1.6.4 Advanced MIMO Techniques

Massive MIMO

Massive MIMO extends the MIMO concept by employing a large number of antennas at the base station, significantly increasing spectral and energy efficiency. However, challenges such as pilot contamination and hardware impairments require innovative solutions.

Multi-User MIMO (MU-MIMO)

MU-MIMO allows a base station to communicate with multiple users simultaneously, increasing system capacity and spectral efficiency. Techniques such as advanced precoding and user scheduling are crucial for its implementation.

1.6.5 Conclusion

MIMO technology has revolutionized wireless communications, enabling higher data rates, improved reliability, and efficient utilization of the spectrum. Its continued evolution and integration into next-generation standards will play a pivotal role in shaping the future of wireless systems.

1.7 Wideband Wireless Channel Estimation

1.7.1 Introduction to CSI and CFR

CSI is a critical concept in modern wireless communication systems, representing the known properties of the channel through which a signal propagates. It encapsulates how a transmitted signal is altered due to multi-path

propagation, scattering, fading, and other environmental effects. CSI enables the optimization of communication systems by adapting transmissions to current channel conditions, improving reliability and achieving higher data rates.

The Channel Frequency Response (CFR) is a frequency-dependent representation of the wireless channel, defined as:

$$H(f;\theta) = \sum_{n=1}^{N} a_n(\theta) e^{-j2\pi f \tau_n(\theta)}, \tag{1.37}$$

where:

- $a_n(\theta)$ represents the amplitude attenuation of the n-th path.
- $\tau_n(\theta)$ denotes the propagation delay of the n-th path.
- f is the carrier frequency.
- N is the number of multi-path components.

CSI incorporates the CFR's amplitude ($|H|$) and phase ($\angle H$) information, reflecting the channel's characteristics, which vary dynamically based on the motion and position of transmitters, receivers, and surrounding objects.

1.7.2 Pilot Sequences for Channel Estimation

Pilot sequences are pre-defined signals known to both the transmitter and the receiver, designed to facilitate channel estimation. These sequences serve as references, allowing the receiver to infer the channel's impact by comparing the received signal with the transmitted one. Pilot sequences are inserted

into the transmitted signal periodically, either in the preamble or within the data stream, as seen in systems like OFDM.

Effective pilot sequences exhibit key properties:

- Orthogonality ensures minimal interference between pilots transmitted from different antennas.
- Strong correlation properties enhance the accuracy of channel estimation by mitigating noise effects.
- They assist in synchronization, aligning timing and frequency between the transmitter and receiver.

In OFDM systems, pilot tones distributed across subcarriers enable the estimation of the CFR for each subcarrier, thereby supporting frequency-selective analysis.

1.7.3 Pilot Placement Strategies

The placement of pilot sequences is crucial for effective channel estimation and depends on the channel's time and frequency selectivity. Common strategies include:

Block-Type Pilots

Block-type pilots occupy all subcarriers in specific OFDM symbols. They are ideal for slowly varying channels where the channel is approximately constant over a block. The receiver uses these pilots to estimate the channel over the entire frequency band.

Comb-Type Pilots

Comb-type pilots are periodically inserted on specific subcarriers across all OFDM symbols. This approach is suited for channels that vary rapidly with time, requiring interpolation techniques such as linear or spline interpolation to estimate the channel for unmeasured subcarriers.

Lattice-Type Pilots

Lattice-type pilots combine the features of block and comb strategies, placing pilots strategically across both time and frequency domains. This method strikes a balance between overhead and estimation accuracy, making it effective for doubly-selective channels.

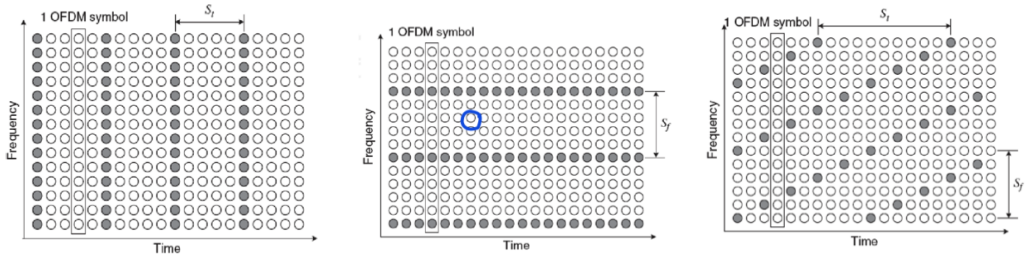


Figure 1.11: Pilot placement strategies: (a) Block-type, (b) Comb-type, and (c) Lattice-type arrangements.

1.7.4 Channel Estimation Techniques

Channel estimation involves determining the CSI by analyzing the received signals, often aided by pilot sequences. The goal is to recover the channel characteristics to optimize transmission. The estimation process can follow two main approaches:

Least Squares (LS) Estimation

The LS method minimizes the error without requiring prior knowledge of the channel or noise statistics. Given a received pilot matrix $\mathbf{Y}$ and a transmitted pilot matrix $\mathbf{P}$, the estimated channel matrix is:

$$\hat{\mathbf{H}}_{\text{LS}} = \mathbf{Y}\mathbf{P}^H(\mathbf{P}\mathbf{P}^H)^{-1}, \tag{1.38}$$

where $\mathbf{P}^H$ is the conjugate transpose of the pilot matrix. The mean squared error (MSE) of the estimation is inversely proportional to $\text{tr}(\mathbf{P}\mathbf{P}^H)$, which can be minimized with an optimized pilot matrix.

Minimum Mean Square Error (MMSE) Estimation

MMSE estimation leverages prior knowledge of the channel and noise statistics to refine the LS estimate. For a Rayleigh fading channel, the MMSE estimate is given by:

$$\hat{\mathbf{H}}_{\text{MMSE}} = \mathbf{R}_{HH}\mathbf{P}^H(\mathbf{P}\mathbf{R}_{HH}\mathbf{P}^H + \sigma_W^2\mathbf{I})^{-1}\mathbf{Y}, \tag{1.39}$$

where:

- $\mathbf{R}_{HH}$ is the channel correlation matrix,
- σ_W^2 is the noise variance,
- $\mathbf{I}$ is the identity matrix.

MMSE provides more accurate channel estimates, particularly in spatially correlated channels, but requires accurate knowledge of channel and noise

statistics.

1.7.5 Conclusion

The effective estimation of CSI is pivotal for modern wireless communication systems. Techniques like LS and MMSE, combined with strategic pilot placement, ensure robust channel characterization, enabling optimized data transmission and enhanced system performance. These methods balance complexity, accuracy, and overhead, making them integral to the design of efficient communication protocols.

1.8 Introduction to 802.11 Frame Structure and Its Relevance in Wi-Fi Networks

Frames are the fundamental units of communication in Wi-Fi networks, facilitating the exchange of data, control signals, and management information between devices. They are utilized to ensure efficient, reliable, and structured communication, enabling tasks such as network configuration, medium access control, and secure data transmission.

The 802.11 standard defines three primary types of frames**management**, **control**, and **data**each serving a specific purpose in the operation of Wi-Fi networks. Frames in Wi-Fi operate primarily at the Data Link Layer of the OSI model, specifically within the MAC (Medium Access Control) sublayer. They handle tasks such as addressing, error detection, and access control for the wireless medium, interfacing directly with the Physical Layer for signal

transmission and reception. These frames form the backbone of communication in wireless systems, enabling tasks such as network management, medium access control, and data transfer. By understanding the structure and utility of these frames, we gain insights into how Wi-Fi networks ensure efficient, reliable, and secure communication.

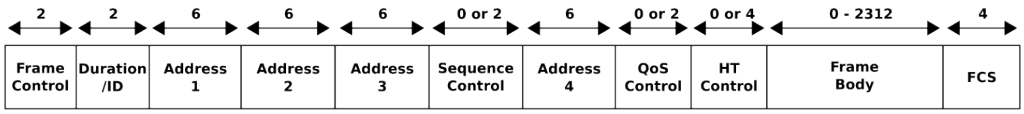


Figure 1.12: Generic 802.11 MAC data frame

1.8.1 Frame Structure and Its Components

Each 802.11 frame comprises three major components:

1. **Header**: Contains critical metadata, including source and destination addresses, frame type, and quality of service (QoS) indicators.

2. **Body**: Encapsulates the higher-layer protocol data (layers 37) or control information, depending on the frame type.

3. **Trailer**: Includes the Frame Check Sequence (FCS), a cyclic redundancy check ensuring the frames integrity during transmission.

The **Frame Control** field in the header dictates the type and subtype of the frame, enabling the differentiation of management, control, and data frames. The **Duration/Connection ID** field manages channel allocation, while address fields ensure correct routing of data packets within the Basic Service Set (BSS).

1.8.2 Frame Types: Functions and Use Cases

Management Frames

- Facilitate BSS operations such as authentication, association, and probing.
- Contain subtypes like **Beacon Frames** (advertising network configurations) and **Authentication Frames** (initiating secure connections).

Control Frames

- Regulate access to the medium and ensure reliable data delivery.
- Examples include **Acknowledgement (ACK)** frames, which confirm receipt of data, and **Request-to-Send (RTS)/Clear-to-Send (CTS)** frames, which manage medium access to avoid collisions.

Data Frames

- Carry higher-layer protocol data within the frame body.
- QoS Data Frames extend standard data frames with priority indicators to optimize performance for latency-sensitive applications like voice or video streaming.

The precise design and functionality of 802.11 frames enable Wi-Fi networks to maintain high performance in dynamic environments. Control mechanisms, such as the **Frame Check Sequence (FCS)**, ensure data

integrity, while QoS features prioritize critical traffic. Additionally, understanding the structure of frames is crucial for tasks like network troubleshooting, optimization, and security analysis.

Table 1.3: Modulation and Coding Schemes (MCS) in IEEE 802.11ax. As detailed in the online reference [1], the full MCS table includes additional parameters such as spatial streams and guard interval variations.

MCS Index	**Modulation**	**Coding Rate**	**Data Rate Efficiency**
0	BPSK	1/2	Low
1	QPSK	1/2	Low
2	QPSK	3/4	Low
3	16-QAM	1/2	Medium
4	16-QAM	3/4	Medium
5	64-QAM	2/3	Medium-High
6	64-QAM	3/4	Medium-High
7	64-QAM	5/6	High
8	256-QAM	3/4	Very High
9	256-QAM	5/6	Very High
10	1024-QAM	3/4	Ultra High
11	1024-QAM	5/6	Ultra High

1.9 Comparison of Wi-Fi Metrics

After introducing the fundamental concepts underlying Wi-Fi communication, this final section focuses on comparing key metrics that characterize the performance and behavior of Wi-Fi systems. Metrics such as throughput, latency, signal strength, and channel utilization are critical for evaluating

network performance in various scenarios. By analyzing these metrics, we gain insights into the trade-offs and limitations of Wi-Fi technologies.

Table 1.4: Overview Comparison of 802.11n, 802.11ac, and 802.11ax

Feature	**802.11n**	**802.11ac**	**802.11ax**
Channel Width (MHz)	20, 40	20, 40, 80, 80+80, 160	20, 40, 80, 80+80, 160
Subcarrier Spacing (kHz)	312.5	312.5	78.125
Symbol Time (μs)	3.2	3.2	12.8
Cyclic Prefix (μs)	0.8	0.8, 0.4	0.8, 1.6, 3.2
MU-MIMO	No	Downlink	Uplink and Downlink
Modulation	OFDM	OFDM	OFDM, OFDMA
Data Subcarrier Modulation	BPSK, QPSK, 16-QAM, 64-QAM	BPSK, QPSK, 16-QAM, 64-QAM, 256-QAM	BPSK, QPSK, 16-QAM, 64-QAM, 256-QAM, 1024-QAM
Coding	BCC (Mandatory), LDPC (Optional)	BCC (Mandatory), LDPC (Optional)	BCC (Mandatory), LDPC (Mandatory)

1.9.1 Throughput vs. Latency

Throughput and latency are among the most critical performance indicators in wireless communication. While throughput measures the maximum data rate achievable over the network, latency quantifies the time delay experienced during data transmission. Wi-Fi generations, from 802.11b to

802.11ax, have seen substantial improvements in throughput, but these gains often come with potential trade-offs in latency due to increased protocol complexity and higher channel utilization.

Table 1.5: Throughput and Latency Comparison Across Wi-Fi Standards

Wi-Fi Standard	**Maximum Throughput (Mbps)**	**Average Latency (ms)**
802.11b	11	20–50
802.11g	54	10–30
802.11n	600	5–15
802.11ac	6933	2–10
802.11ax	9608	<5

1.9.2 Signal Strength and Coverage

Signal strength, typically represented as RSSI (Received Signal Strength Indicator), plays a vital role in determining the quality of a Wi-Fi connection. It is influenced by environmental factors such as obstacles, interference, and distance from the access point. As shown in Table **??**, Wi-Fi operates across multiple frequency bands, with higher frequencies offering faster speeds but shorter range.

Table 1.6: Signal Strength and Coverage Characteristics

Frequency Band	**Range (Meters)**	**Max Channels**	**Signal Penetration**
2.4 GHz	100–150	3 Non-overlapping	High
5 GHz	50–100	23 Non-overlapping	Medium
6 GHz (Wi-Fi 6E)	30–50	59 Non-overlapping	Low

1.9.3 Channel Utilization and Interference

Channel utilization is a metric that reflects how efficiently the available spectrum is being used. Wi-Fi networks in densely populated areas often face interference from overlapping channels, which can degrade performance. The introduction of OFDMA in 802.11ax has significantly improved channel efficiency by allowing multiple users to share the spectrum dynamically.

Table 1.7: Channel Utilization Efficiency Across Wi-Fi Standards

Wi-Fi Standard	**Channel Width (MHz)**	**Max Subcarriers**	**Efficiency**
802.11n	20/40	64	Moderate
802.11ac	20/40/80/160	256	High
802.11ax	20/40/80/160	1024	Very High

1.9.4 Summary of Metrics

The tables presented in this section highlight the evolution of Wi-Fi technology, emphasizing improvements in throughput, efficiency, and scalability. These advancements demonstrate how Wi-Fi continues to adapt to increasing demands for speed, reliability, and user density, paving the way for enhanced user experiences in both personal and enterprise networks.

Chapter 2

Reinforcement Learning: The Decision Making Engine of Machine Learning

2.1 Introduction

Reinforcement Learning (RL) is a powerful machine learning paradigm that enables agents to make sequential decisions by interacting with an environment and optimizing their behavior based on rewards. Unlike supervised learning, where labeled data guides model training, RL agents learn autonomously through trial and error, refining their strategies over time. This ability to learn optimal policies from experience has made RL a fundamental tool in solving complex decision-making problems across various domains, including robotics, autonomous systems, and telecommunications.

This chapter provides a comprehensive introduction to reinforcement

learning, detailing its theoretical foundations and core principles. It begins with an overview of the MDP, the mathematical framework underlying RL, followed by an exploration of key components such as states, actions, rewards, and policies. Subsequently, we examine classical and modern RL algorithms, emphasizing their practical applications and limitations. Special attention is given to value-based and policy-based methods, with an in-depth discussion on policy optimization techniques.

Furthermore, this chapter introduces PPO, a state-of-the-art RL algorithm that balances exploration and exploitation while ensuring stable policy updates. The theoretical derivation of PPO is presented, illustrating its evolution from fundamental policy gradient methods and trust-region optimization techniques. By establishing a rigorous understanding of these concepts, this chapter serves as a foundation for the RL methodologies employed in this thesis, particularly in the context of intelligent decision-making for antenna selection in WiFi networks.

2.2 Machine Learning: An Overview

Machine Learning (ML) is a subfield of artificial intelligence (AI) that focuses on developing algorithms capable of learning patterns and making decisions or predictions based on data. Unlike traditional programming, where explicit instructions dictate the system's behavior, machine learning relies on data-driven approaches to infer patterns and relationships, allowing systems to adapt and improve over time.

Figure 2.1: Comparison between Traditional programming approach and machine learning: (a) Traditional programming framework, (b) Generic machine learning framework.

The rise of machine learning is fueled by the exponential growth in data availability, increased computational power, and advancements in algorithmic research. ML has found applications across diverse fields, including healthcare, finance, marketing, robotics, and engineering. This section provides an overview of the three primary categories of machine learning: supervised, unsupervised, and reinforcement learning, discussing their distinct characteristics, applications, and use cases.

2.2.1 When to Use Machine Learning

Machine learning is particularly useful in scenarios where traditional programming approaches fall short due to complexity or variability. Some typical scenarios include:

- **Data-Driven Problems:** When large datasets exist but explicit rules to solve the problem are difficult to define.
- **Dynamic Systems:** When the problem environment is dynamic, and the system needs to adapt to changing conditions.

- **Pattern Recognition:** For identifying patterns in high-dimensional data, such as image classification or speech recognition.
- **Automation and Prediction:** When predictive capabilities are required, such as forecasting stock prices or automating decision-making.

In these cases, ML provides a robust approach for extracting insights, making predictions, or optimizing systems.

2.2.2 Categories of Machine Learning

Machine learning is broadly categorized into three main types: supervised learning, unsupervised learning, and reinforcement learning. Each type addresses specific classes of problems and employs unique methodologies.

Supervised Learning

Supervised learning involves training a model on labeled data, where each input is paired with a corresponding output. The algorithm learns a mapping function between inputs and outputs by minimizing the error between predicted and actual results. The two primary tasks in supervised learning are:

- **Regression:** Predicting continuous values. Examples include forecasting temperatures, predicting housing prices, or estimating sales trends.
- **Classification:** Assigning inputs to discrete categories. Common examples include spam email detection, image classification, and medical diagnosis.

Supervised learning is ideal when labeled data is abundant, and the goal is to make predictions or categorize new, unseen inputs.

Unsupervised Learning

Unsupervised learning operates on unlabeled data, aiming to uncover hidden patterns or structures within the dataset. The algorithm explores the data's intrinsic properties without predefined outcomes, making it suitable for tasks such as:

- **Clustering:** Grouping similar data points together based on their features. Applications include customer segmentation, social network analysis, and anomaly detection.
- **Dimensionality Reduction:** Simplifying data by reducing the number of features while retaining essential information. Principal Component Analysis (PCA) is a common technique used for data visualization and preprocessing.

Unsupervised learning is particularly valuable in exploratory data analysis, where the goal is to gain insights or discover patterns in complex datasets without prior knowledge of the relationships.

Reinforcement Learning

RL involves training an agent to make a sequence of decisions by interacting with an environment. The agent learns by receiving rewards or penalties for its actions, optimizing its policy to maximize cumulative rewards over time. This approach will be deeply analyzed in the following section.

2.2.3 Choosing the Right Approach

Selecting the appropriate machine learning approach depends on the nature of the problem and the availability of data:

- **Supervised Learning:** Use when labeled data is available, and the goal is to predict outcomes or classify inputs.
- **Unsupervised Learning:** Use when data lacks labels, and the objective is to explore or model the data's underlying structure.
- **Reinforcement Learning:** Use when the problem involves sequential decision-making, and a clear reward mechanism exists to guide the learning process.

Understanding these categories and their applications is fundamental to leveraging machine learning effectively to solve real-world problems.

2.3 Markov Decision Process and Reinforcement Learning

RL is a fundamental branch of machine learning, focused on training agents to make sequential decisions in dynamic environments. The goal of this chapter is to provide a foundational understanding of RL concepts and algorithms, equipping the reader with the necessary background to comprehend the methodologies implemented in this thesis. While this chapter focuses on establishing a theoretical foundation, the specific problem modeling and

its application within the RL framework are deferred to Chapter 3. RL operates within the paradigm of learning through interaction, where an agent learns to achieve a predefined goal by exploring an environment and receiving feedback in the form of rewards. The learning process is centered around discovering a policy that maps states of the environment to optimal actions, balancing the trade-off between exploration (discovering new strategies) and exploitation (leveraging known strategies). This chapter will first outline key concepts such as the MDP, the mathematical framework underpinning RL, and introduce essential components like states, actions, rewards, and policies. Subsequently, a review of classical and modern RL algorithms will be presented, with a particular emphasis on those utilized in this thesis. For detailed derivations and additional foundational mathematics, the reader is referred to the Appendix 5.0.5, where such discussions are provided for completeness without interrupting the main narrative flow. Through this structured approach, this chapter aims to establish a solid understanding of RL principles while setting the stage for the practical implementations and problem formulations discussed in the subsequent chapters.

2.3.1 Markov Decision Process

In mathematics, a MDP is a discrete-time stochastic control process. Formally, an MDP describes a fully observable environment for RL. Theoretical results in RL are based on MDP, meaning that if the problem is formulated as an MDP, the RL approach can be used to find solutions. Specifically, RL is a solution approach for MDP problems when the parameters of the

MDP model are unknown. To understand MDP, we first present the Markov property, i.e., "the future is independent of the past given the present" [2,3]. Once the current state is known, the information gathered from old experiences may not be necessary, and it is a sufficient statistic to predict the next action to take [3]. Mathematically, a state s_t at time t has the Markov property if and only if:

$$P(s_t|s_{t-1}) = P(s_t|s_{t-1}, s_{t-2} \ldots, s_1). \tag{2.1}$$

An MDP models an environment in which all states have the Markov property. Formally, an MDP is defined as a tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where:

- $\mathcal{S}$ is the state space : it refers to the representation of the current situations or conditions of the environment in which an agent operates.
- $\mathcal{A}$ is the action space.
- P is a state transition probability function $P : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$.
- r is a reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$.
- $\gamma \in [0, 1]$ is the discount factor representing the trade-off between immediate and upcoming rewards.

The full observability assumption allows the agent to have a full observation of the current state $s_t \in \mathcal{S}$ in every time step t. Given the state s_t, the agent makes a decision of taking the action a_t and observes the new state s' in the MDP environment, where s' is sampled from the transition probability $P(\cdot|s, a)$, and an immediate reward $r(s, a, s')$ is provided. Hence, the

expected return is expressed as:

$$\mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t r(s,a,s')|a\sim\pi(\cdot|s),s_0\right], \tag{2.2}$$

which is also referred to as the infinite-horizon discounted return. Alternatively, for episodic tasks, the finite-horizon discounted return is expressed as:

$$\mathbb{E}\left[\sum_{t=0}^{H}\gamma^t r(s,a,s')|a\sim\pi(\cdot|s),s_0\right], \tag{2.3}$$

where the return is computed over a horizon H. The principal goal of RL algorithms is to learn a policy that maximizes the expected return.

2.3.2 Partially Observable Markov Decision Process

In many real-world scenarios, the assumption of full state observability in MDPs is unrealistic. IoT devices, for example, often collect noisy and limited measurements of their surroundings. To address these limitations, Partially Observable Markov Decision Processes (POMDPs) provide a framework for modeling environments where an agent's access to the state information is restricted or noisy.

Definition

Formally, a POMDP is defined as a 7-tuple $(\mathcal{S},\mathcal{A},T,r,\gamma,\mathcal{O},O)$, where:

- $\mathcal{S}$: A finite set of states.
- $\mathcal{A}$: A finite set of actions.

- $P(s' \mid s, a)$: The state transition probability, representing the likelihood of transitioning to state $s' \in \mathcal{S}$ from state $s \in \mathcal{S}$ after taking action $a \in \mathcal{A}$.
- $r(s, a)$: The reward function, which provides the immediate reward for being in state $s \in \mathcal{S}$ and taking action $a \in \mathcal{A}$.
- $\gamma \in [0, 1)$: The discount factor that determines the importance of future rewards.
- $\mathcal{O}$: The set of observations available to the agent.
- $O(o \mid s', a)$: The observation probability, representing the likelihood of observing $o \in \mathcal{O}$ given the resulting state $s' \in$ and action $a \in \mathcal{A}$.

At each time step, the agent takes an action $a \in \mathcal{A}$, causing the environment to transition to a new state $s' \in \mathcal{S}$ with probability $P(s' \mid s, a)$. The agent receives a reward $r(s, a)$ and an observation $o \in \mathcal{O}$, which provides partial information about the new state s'.

State Estimation and Belief Representation

Since the true state s is not directly observable, the agent must estimate its current state. Two common approaches are:

1. **History-Based Methods:** Estimating the current state using the history of past observations and actions.
2. **Belief State Representation:** Maintaining a belief state $b(s)$, which is a probability distribution over all possible states $s \in \mathcal{S}$. The belief

state is updated after each action and observation using:

$$b'(s') = \eta O(o \mid s', a) \sum_{s \in \mathcal{S}} P(s' \mid s, a) b(s), \tag{2.4}$$

where η is a normalizing constant given by:

$$\eta = \frac{1}{\Pr(o \mid b, a)}, \quad \text{with } \Pr(o \mid b, a) = \sum_{s' \in \mathcal{S}} O(o \mid s', a) \sum_{s \in \mathcal{S}} P(s' \mid s, a) b(s).$$

Belief MDP Formulation

To address the challenges posed by partial observability, a POMDP can be reformulated as a belief-based MDP. In this formulation, the belief state $b(s)$ serves as the fully observable "state" in the reformulated problem.

The belief MDP is defined as a tuple $(B, \mathcal{A}, \tau, r, \gamma)$, where:

- B: The set of belief states, representing probability distributions over $\mathcal{S}$.
- $\mathcal{A}$: The same set of actions as the original POMDP.
- τ: The belief state transition function, as defined by the belief update equation:

$$\tau(b, a, o) = \frac{O(o \mid s', a) \sum_{s \in \mathcal{S}} P(s' \mid s, a) b(s)}{\sum_{s' \in \mathcal{S}} O(o \mid s', a) \sum_{s \in \mathcal{S}} P(s' \mid s, a) b(s)}. \tag{2.5}$$

- $r(b, a)$: The reward function on belief states, computed as:

$$r(b, a) = \sum_{s \in \mathcal{S}} b(s) r(s, a). \tag{2.6}$$

- γ: The discount factor, identical to the original POMDP.

2.3.3 Relationship Between MDP and POMDP Solving Strategies

The strategies for solving a POMDP closely resemble those used for MDPs, with the critical difference being that POMDPs operate on the belief state rather than the actual state. This reformulation allows us to leverage MDP-solving methods by treating the belief state as the fully observable state in a belief-based MDP. In the following sections, we will introduce methods to solve MDPs and POMDPs. However, due to the equivalence in solving strategies, we will focus our discussion on MDPs for simplicity.

Comparison to MDPs

POMDPs provide several advantages over MDPs in scenarios with:

- **Noisy Observations:** Sensors provide incomplete or inaccurate information about the state.
- **Hidden States:** Critical system variables are not directly observable.
- **Sequential Decision-Making:** Actions depend on the history of observations, which is naturally incorporated into the belief state.

In conclusion, POMDPs offer a robust framework for decision-making under uncertainty, extending the utility of MDPs to a wider range of real-world applications. By reformulating the problem as a belief-based MDP, standard

solution techniques can be leveraged, albeit with additional computational complexity.

2.4 Solving MDP Problems: Policies and Value Functions

To solve an MDPs, the goal is to maximize the expected *return*, which represents the cumulative reward collected over time. The return serves as a measure of the agent's performance in the environment and is formally defined as:

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1}, \tag{2.7}$$

where:

- r_{t+k+1} is the immediate reward received at time $t + k + 1$,
- $\gamma \in [0, 1]$ is the discount factor, which determines the importance of future rewards relative to immediate rewards,
- T is the time horizon, which can be finite (episodic tasks) or infinite (continuing tasks).

The discount factor γ plays a critical role in shaping the agent's behavior:

- A lower γ emphasizes immediate rewards, leading to a more greedy behavior.

- A higher γ values future rewards more, encouraging the agent to plan over a longer time horizon.

- For episodic tasks, γ can be set to 1 if there is a well-defined terminal state, whereas for continuing tasks, γ must be strictly less than 1 to ensure convergence of the return.

2.4.1 Policies

The agents behavior is governed by a *policy*, which specifies the actions to take in each state:

- **Deterministic Policy:** A function $\pi(s) = a$ maps a state $s \in \mathcal{S}$ to a specific action $a \in \mathcal{A}$.

- **Stochastic Policy:** A probability distribution $\pi(a|s) \in [0, 1]$ returns the likelihood of taking action a in state s.

2.4.2 Trajectories

A trajectory τ is a sequence of states and actions in the environment:

$$\tau = (s_0, a_0, s_1, a_1, \dots). \tag{2.8}$$

The very first state of the environment, s_0, is randomly sampled from the start-state distribution, sometimes denoted by ρ_0:

$$s_0 \sim \rho_0(\cdot). \tag{2.9}$$

State transitions, which describe how the environment evolves between the state at time t, s_t, and the state at $t+1$, s_{t+1}, are governed by the natural laws of the environment and depend only on the most recent action, a_t. These transitions can be either deterministic:

$$s_{t+1} = f(s_t, a_t), \tag{2.10}$$

or stochastic:

$$s_{t+1} \sim P(\cdot|s_t, a_t). \tag{2.11}$$

Actions are chosen by an agent according to its policy.

2.4.3 Value Functions

To evaluate the quality of a policy, value functions are used. These functions estimate the expected return under a given policy:

- **State-Value Function $V_\pi(s)$:** The expected return starting from state s and following policy π:

$$V_\pi(s) = \mathbb{E}_\pi \left[R_t \,|\, s_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{T} \gamma^k r_{t+k+1} \,|\, s_t = s \right]. \tag{2.12}$$

- **State-Action Value Function $Q_\pi(s, a)$:** The expected return starting from state s, taking action a, and then following policy π:

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[R_t \,|\, s_t = s, a_t = a \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{T} \gamma^k r_{t+k+1} \,|\, s_t = s, a_t = a \right]. \tag{2.13}$$

The relationship between $V_\pi(s)$ and $Q_\pi(s,a)$ is expressed as:

$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) Q_\pi(s,a). \tag{2.14}$$

2.4.4 Optimal Policy and Value Functions

The ultimate objective is to find the *optimal policy* π^* that maximizes the expected return. The corresponding optimal value functions are:

- **Optimal State-Value Function $V^*(s)$:**

$$V^*(s) = \max_\pi V_\pi(s), \tag{2.15}$$

  which represents the highest expected return achievable from state s.

- **Optimal State-Action Value Function $Q^*(s,a)$:**

$$Q^*(s,a) = \max_\pi Q_\pi(s,a), \tag{2.16}$$

  which represents the highest expected return achievable from state s by taking action a and then following the optimal policy.

The optimal policy can be directly derived from $Q^*(s,a)$:

$$\pi^*(s) = arg\,\max_{a \in A} Q^*(s,a) \tag{2.17}$$

This framework establishes the foundation for reinforcement learning, where algorithms iteratively approximate the optimal policy and value functions, even when the dynamics of the environment are unknown. Further

details on specific solution methods, such as policy iteration and value iteration, are provided in the appendix A.

2.4.5 Exploration and Exploitation in RL

One of the central challenges in RL is the trade-off between exploration and exploitation. This trade-off arises because an agent must balance two competing objectives: (1) exploiting its current knowledge to maximize immediate rewards and (2) exploring the environment to gather more information that might lead to higher rewards in the future. This fundamental paradigm is crucial for the agent's ability to learn an optimal policy over time.

Exploration

Exploration refers to the agent's actions aimed at gaining new knowledge about the environment. By trying actions that it has not taken frequently or at all, the agent can discover new states and potentially better reward paths. Exploration is particularly important in the early stages of learning when the agent's knowledge of the environment is incomplete or inaccurate. Key aspects of exploration include:

- **Randomness in Action Selection:** The agent deliberately chooses actions that it believes might not yield the highest immediate reward but could provide information about the environment.
- **Stochastic Policies:** Policies that allow the agent to sample actions probabilistically rather than deterministically, enabling exploration even in familiar states.

- **Role in Exploration:** Exploration ensures that the agent does not become trapped in suboptimal solutions due to insufficient knowledge of the environment. It allows the agent to evaluate alternative strategies and adjust its policy accordingly.

Exploitation

Exploitation, on the other hand, involves selecting actions that maximize the agent's immediate expected reward based on its current policy. This approach leverages the agent's accumulated knowledge to achieve the best performance in the short term. Key aspects of exploitation include:

- **Greedy Action Selection:** The agent chooses the action with the highest estimated value or reward in its current knowledge base.
- **Short-Term Performance:** Exploitation ensures that the agent performs well in the present by taking actions that are likely to yield high rewards.
- **Limitations:** Over-reliance on exploitation can lead to suboptimal policies, as the agent may overlook better actions or strategies that require initial exploration to uncover.

The Trade-Off

The balance between exploration and exploitation is critical for the success of reinforcement learning. Too much exploration can lead to wasted time and effort in actions that yield little reward, while excessive exploitation can cause the agent to miss out on potentially better long-term strategies.

The trade-off is often managed using strategies that encourage exploration initially and gradually shift toward exploitation as the agent gains confidence in its policy.

Several approaches are commonly used to manage this trade-off:

- **ϵ-Greedy Policies:** The agent selects a random action with probability ϵ (exploration) and the best-known action with probability $1 - \epsilon$ (exploitation). The value of ϵ can decay over time, allowing the agent to explore more initially and exploit as it learns.

- **Stochastic Policy Sampling:** Actions are sampled from a probability distribution derived from the policy network. The probability of selecting an action is proportional to its assigned weight, ensuring that higher-value actions are more likely to be chosen while still allowing for occasional exploration.

- **Uncertainty-Based Exploration:** Techniques such as Upper Confidence Bound (UCB) encourage actions with high uncertainty in their value estimates, balancing exploration and exploitation.

- **Intrinsic Motivation:** The agent is rewarded for exploring novel states or reducing uncertainty in its knowledge.

The Role of Exploration and Exploitation in RL

The exploration-exploitation dilemma is not unique to reinforcement learning but is particularly pronounced in RL due to the sequential decision-making nature of the problem. The agent must consider not only the immediate

rewards of its actions but also their long-term impact on its understanding of the environment and future performance.

Effective management of this trade-off is essential for discovering optimal policies, particularly in environments with sparse or deceptive rewards. For instance, an agent may need to take seemingly suboptimal actions to transition to states that yield higher rewards in the long run. Conversely, failure to exploit known rewards efficiently can delay convergence to the optimal policy.

Challenges and Open Questions

Balancing exploration and exploitation effectively is a non-trivial challenge, particularly in complex or dynamic environments. Key questions include:

- How can the agent adapt its exploration strategy as it learns more about the environment?
- What are the best methods for quantifying the trade-off in different environments or problem settings?
- How can exploration be guided or prioritized to focus on regions of the state space that are most likely to yield valuable information?

In summary, the exploration-exploitation paradigm is a cornerstone of reinforcement learning, enabling agents to balance immediate performance with long-term learning. Its effective management is critical for developing robust and adaptable RL algorithms.

2.4.6 Learning Paradigms: Policy Dependence and Data Interaction

A fundamental aspect of reinforcement learning is the way an agent collects and utilizes experiences to improve its decision-making process. Different strategies define how policies are updated and how data is leveraged, leading to two key distinctions: on-policy versus off-policy learning and online versus offline learning.

An on-policy algorithm learns exclusively from data collected while following the current policy. This ensures that the learned policy remains consistent with the agents recent experiences. However, it can be data inefficient, as previously collected experiences become less useful once the policy changes. Algorithms such as PPO and State-Action-Reward-State-Action (SARSA) fall within this category.

In contrast, off-policy learning allows an agent to improve a policy using data collected under different policies, including past iterations or even external sources. This facilitates more effective data reuse, leading to improved sample efficiency and stability. Algorithms such as Q-learning and Deep Q-Networks (DQN) exploit this characteristic, often incorporating an experience replay buffer to store and revisit past interactions.

Another critical distinction concerns the manner in which policy updates occur. In online learning, the agent updates its model incrementally while interacting with the environment. This approach is well-suited for dynamic and evolving scenarios, such as robotic control or adaptive network management, where continuous learning is essential.

Conversely, offline learning relies on a fixed dataset of pre-collected experiences, without the possibility of further interaction with the environment during training. This paradigm is particularly useful in applications where data acquisition is costly or unsafe, such as autonomous driving or medical diagnosis. However, it requires sophisticated techniques to generalize effectively from limited data and to mitigate distributional shift, which arises when the data used for training differs from real-world deployment scenarios.

The choice between these learning paradigms depends on the specific application and objectives of the reinforcement learning system. On-policy learning is often preferred for ensuring stability in learned policies, whereas off-policy learning maximizes data efficiency. Similarly, online learning enables rapid adaptation, while offline learning is crucial when real-time data collection is impractical or expensive.

Recent advances in reinforcement learning seek to integrate these paradigms to balance exploration, exploitation, and data efficiency. Hybrid approaches, such as offline-to-online reinforcement learning, aim to combine the benefits of both strategies by pretraining policies on static datasets before fine-tuning them through real-time interactions. Addressing these trade-offs remains a central challenge in developing robust and scalable reinforcement learning algorithms.

2.4.7 Illustrative Example of a Markov Decision Process

To provide an intuitive understanding of MDPs, Figure 2.2 presents a simple example consisting of three states, possible actions, stochastic transitions, and associated rewards. This visualization serves as a reference for how an agent interacts with its environment within an MDP framework.

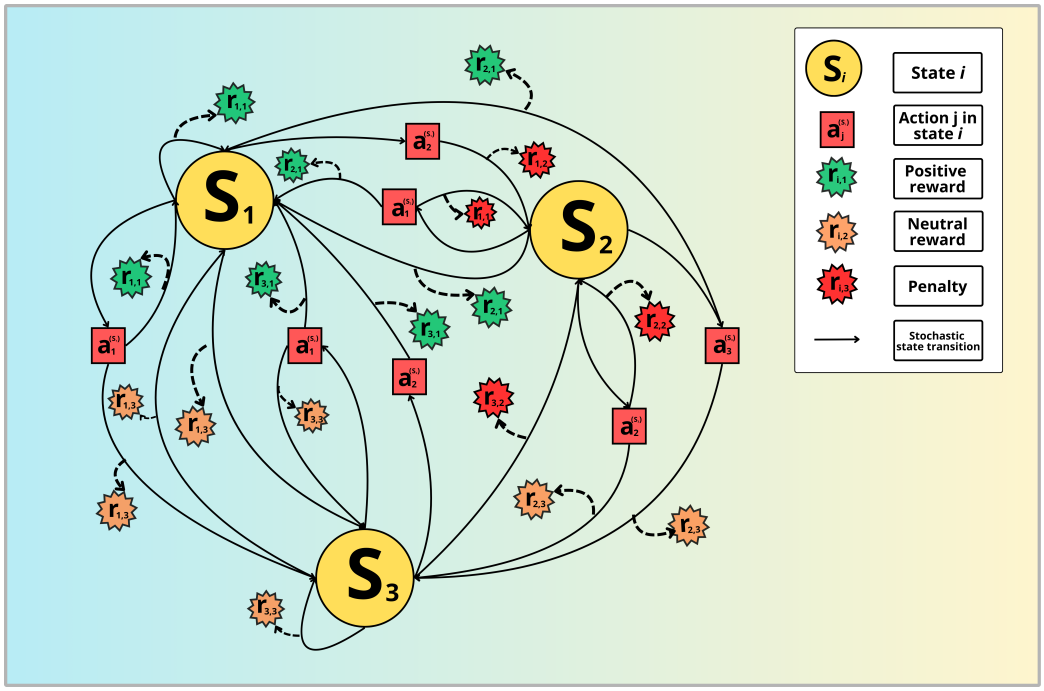


Figure 2.2: Example of a Markov Decision Process (MDP) with three states, three possible actions per state, and stochastic transitions.

In this example, the system can exist in one of three states, denoted as S_1, S_2, and S_3, each representing different conditions of a communication channel:

- S_1 corresponds to an optimal state with high signal-to-noise ratio (SNR), ensuring the best communication conditions.

- S_3 represents an intermediate state with moderate SNR.
- S_2 denotes the worst condition, where the channel quality is significantly degraded.

At each decision step, the agent selects an action $a_j^{(s)}$ (represented by red squares), which is specific to the state it currently occupies. The policy governing action selection is stochastic, meaning that the same state may lead to different actions with certain probabilities. Once an action is executed, the system undergoes a transition to a new state according to a predefined probability distribution.

However, the evolution of the system is not solely determined by the agents actions. The environment itself introduces variability, causing transitions between states independently of the agents choices. For example, in a wireless communication scenario, factors such as interference, user mobility, or changing environmental conditions may alter the channel state unpredictably. This reflects the dynamic nature of real-world systems, where external influences impact decision-making.

State Transitions and Rewards

Each transition between states is associated with a reward, which is a numerical value reflecting the desirability of reaching a particular state. In this example, the reward is determined solely by the final state after the transition:

- *High reward* (green, $r_{i,1}$) is assigned when the agent reaches or remains in S_1, representing favorable channel conditions.

- *Moderate reward* (orange, $r_{i,2}$) is given for reaching or staying in S_3, corresponding to an average-quality channel.

- *Negative reward* (red, $r_{i,3}$) is received when the system moves into S_2, indicating a poor communication state.

Relevance to Antenna Selection in WiFi Networks

This MDP representation aligns with the problem of antenna selection in WiFi APs, which will be further discussed in Chapter 3. The AP, acting as an intelligent agent, can switch between different antennas in an attempt to optimize network performance. Each antenna selection corresponds to an action, which influences the probability of transitioning to a better or worse channel state. However, external conditions, such as interference from other devices or the movement of a connected user, also contribute to state transitions that are beyond the APs control.

By leveraging reinforcement learning, the AP can learn an optimal policy to maximize long-term performance, aiming to maintain the best possible channel conditions despite the uncertainties introduced by the environment. If the system transitions to a poor state due to external factors, the agent can attempt to mitigate the degradation by selecting a different antenna, increasing the chances of recovering a favorable communication condition.

This example serves as a foundational illustration of how decision-making in uncertain environments can be modeled using MDPs and optimized through reinforcement learning.

2.5 Deep Reinforcement Learning Models

RL problems often involve high-dimensional state spaces, where each state can be represented by a large number of features. In such scenarios, explicitly storing and processing all possible state-action pairs becomes impractical. Traditional RL methods, introduced in the appendix A, rely on tabular approaches, where models such as $V_\pi(s)$ or $Q_\pi(s, a)$ are stored in a table containing the value of each state or the Q-value of every state-action pair. However, these tabular methods are only feasible when dealing with small state and action spaces. Their applicability is severely limited for problems with large or continuous state spaces, as the number of state-action pairs grows exponentially, making memory and computational requirements unmanageable. Moreover, for these methods to converge, every state-action pair must be visited multiple times, which is rarely achievable in complex environments.

A more effective approach is to generalize the learned Q-values by training on a subset of state-action pairs and inferring the values for newly encountered pairs based on their proximity to known ones. This challenge, known as the generalization problem, focuses on transferring acquired knowledge to unseen data, enabling RL algorithms to scale beyond tabular methods and operate effectively in high-dimensional spaces.

2.5.1 Value-Based DRL Methods

Function approximators offer an effective solution to this issue. Instead of storing Q-values in a table, they are represented as a parameterized function

$Q_\theta(s, a)$. This function approximates the optimal function $Q^*(s, a)$ and is optimized iteratively until convergence. Although neural networks are the focus of our discussion as function approximators, other approaches, such as linear models, radial-basis function networks, and support vector regression (SVR), can also be effective. When deep neural networks (DNNs) are used as function approximators, the resulting RL models/methods are referred to as deep reinforcement learning (DRL) models/methods.

Value-based methods represent one of the two primary categories of model-free RL techniques, alongside policy-based methods. In value-based approaches, an approximation of the value function $\hat{V}$ is learned, and the policy is derived by acting greedily with respect to $\hat{V}$. In DRL, function approximators are typically implemented as neural networks parameterized by θ, representing the weights of the connections between layers. Action selection is commonly performed using ϵ-greedy or softmax strategies, making the policy directly dependent on the Q-values generated by the function approximator $Q_\theta(s, a)$ and, consequently, on the weights θ. Thus, the policy is denoted as π_θ. Mathematically, learning is formulated as a minimization problem of a loss function, which is defined based on the parameterized function $Q_\theta(s, a)$ and the real returns R_t sampled from the environment as follows:

$$\mathcal{L}_Q(\theta) = \mathbb{E}_\pi[(R_t - Q_\theta(s, a))^2]. \tag{2.18}$$

Similarly, the value function loss is defined as:

$$\mathcal{L}_V(\phi) = \mathbb{E}_{\pi_\theta}\left[(V_\phi(s) - G_t)^2\right], \tag{2.19}$$

where G_t is the return, which may be defined as $G_t = R_t$ in simple cases or as a bootstrapped estimate, such as $G_t = R_t + \gamma V_\phi(s')$, depending on the specific algorithm. For further details and a more in-depth discussion on value-based methods, see Appendix B

2.5.2 Policy-Gradient Methods

In this section, we discuss policy search methods in DRL that focus on learning the policy π_θ directly, typically represented by a neural network with parameters θ. The learning goal is to maximize an objective function that represents the return over a trajectory. Mathematically, the objective function is defined as follows:

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta}\left[R(\tau)\right] = \mathbb{E}_{\tau \sim p_\theta}\left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t, s_{t+1})\right], \tag{2.20}$$

where $\tau = (s_0, a_0, s_1, a_1, \ldots, s_T, a_T)$ is a trajectory and p_θ is the trajectory distribution. The likelihood of sampling a trajectory τ can be computed as:

$$p_\theta(\tau) = p_0(s_0) \prod_{t=0}^{T} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t), \tag{2.21}$$

where $p_0(s_0)$ defines the probability of starting from state s_0 and $p(s_{t+1}|s_t, a_t)$ is the transition probability defining the Markov Decision Process (MDP). However, since we do not have knowledge of the system dynamics, we do not use this term to define the objective function. Instead, we estimate the objective function $J(\theta)$ using trajectories sampled from p_θ. Using the Monte Carlo method A.2.2, we approximate the objective as follows:

$$J(\theta) = \mathbb{E}_{r\sim p_\theta(\tau)}\left[\sum_t r(s_t, a_t)\right] \approx \frac{1}{N}\sum_{i=1}^{N}\sum_t r(s_t, a_t). \tag{2.22}$$

This approximation relies on generating multiple trajectories $\{\tau_i\}$ by running the policy and averaging the trajectory rewards. The more trajectories are generated, the higher the accuracy in evaluating the objective function.

To learn a policy that maximizes the objective function $J(\theta)$, we apply gradient ascent on the weights θ. Therefore, we require the gradient of the objective function with respect to the weights:

$$\nabla_\theta J(\theta) = \frac{\partial J(\theta)}{\partial \theta}. \tag{2.23}$$

Then, the update rule for the weights is given by:

$$\theta \leftarrow \theta + \eta \nabla_\theta J(\theta). \tag{2.24}$$

REINFORCE Algorithm

The REINFORCE algorithm was proposed in 1992 by Williams [4], which estimates the policy gradient using the Monte Carlo method. In this section, we introduce the mathematical derivations that lead to the REINFORCE algorithm and discuss techniques used to reduce the variance in estimating the policy gradient.

Policy Gradient Estimation

Denoting the trajectory reward by $R(\tau) = \sum_{t=1}^{T} r(s_t, a_t)$, the objective function can be rewritten as:

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[R(\tau) \right] = \int p_\theta(\tau) R(\tau) d\tau. \tag{2.25}$$

Applying the log-trick to $\nabla_\theta p_\theta(\tau)$, we obtain:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\nabla_\theta \log p_\theta(\tau) R(\tau) \right]. \tag{2.26}$$

Expanding $\log p_\theta(\tau)$ using Equation (3.18), we get:

$$\nabla_\theta \log p_\theta(\tau) = \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t | s_t). \tag{2.27}$$

Thus, the gradient can be rewritten as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t | s_t) \right) \left(\sum_{t=1}^{T} r(s_t, a_t) \right) \right]. \tag{2.28}$$

Using Monte Carlo sampling, the policy gradient estimate becomes:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t} | s_{i,t}) \right) \left(\sum_{t=1}^{T} r(s_{i,t}, a_{i,t}) \right). \tag{2.29}$$

The policy is then updated as:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta). \tag{2.30}$$

The basic steps of the REINFORCE algorithm can be summarized as follows:

- **Trajectory sampling:** Run the policy $\pi_\theta(\cdot)$ for multiple trajectories $\{\tau_i\}$.

- **Estimation of the policy gradient:** Evaluate $\nabla_\theta J(\theta)$ using Monte Carlo sampling (Equation 3.28).

- **Policy improvement:** Compute new weights θ using gradient ascent (Equation 3.29).

The REINFORCE algorithm is simple but suffers from high variance. Several techniques, such as baselines and advantage functions, can be used to reduce this variance, leading to more stable learning.

Reducing Variance

Policy gradient methods are often inefficient in terms of sample usage when computing the gradient $\nabla_\theta J(\theta)$. In Eq. (2.29), the gradient is estimated via Monte Carlo by executing multiple trajectories. Obtaining an accurate gradient estimate requires a significant number of sampled trajectories, leading to considerable computational expense. Moreover, each trajectory involves summing the rewards across multiple steps, further increasing computational complexity. Another issue with policy gradient methods is their slow convergence, as they demand a substantial amount of samples for policy updates.

Aside from computational inefficiencies, policy gradient methods are also affected by high variance. This issue becomes evident in large-scale applications such as learning resource allocation in an unfamiliar environment or training an agent for beamforming in extensive networks. When sampling

trajectories from an untrained policy, the observed behaviors can vary significantly. To effectively learn an optimal policy, the agent must explore diverse actions across different states, which, in real-world environments, results in an exponentially large search space. In continuous state-action spaces, visiting all action-state pairs is computationally infeasible. Monte Carlo estimation thus presents a trade-off between computational tractability and gradient accuracy.

Baseline A key characteristic of policy gradient descent is its reliance on the trajectory reward:

$$R(\tau) = \sum_t r(a_t, s_t). \tag{2.31}$$

If the reward for a trajectory $R(\tau_1)$ is negative, the policy π_θ is adjusted in the opposite direction of the gradient, reducing the likelihood of selecting that trajectory, i.e., $\pi_\theta(\tau_1) < 0$. Conversely, if the reward is positive, $R(\tau_2) > 0$, the probability $\pi_\theta(\tau_2)$ increases, reinforcing the selection of that trajectory. Introducing a constant b into the trajectory rewards, such that $R(\tau) + b > 0$, can inadvertently increase the probability of suboptimal trajectories. This highlights the sensitivity of policy gradients to shifts and scaling in the reward function. To address this issue, an optimal baseline b is sought to minimize the variance of policy gradient. Including the baseline, the policy gradient takes the form:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla \log \pi_\theta(\tau_i)(R(\tau_i) - b). \tag{2.32}$$

A natural choice for b is the mean reward over all trajectories:

$$b = \frac{1}{N} \sum_{i=1}^{N} R(\tau_i), \tag{2.33}$$

so that high-reward trajectories are more frequently selected while low-reward ones are discouraged. This leads to two crucial questions: (i) Can the policy gradient be expressed as in Eq.(2.32)? (ii) If so, what is the optimal b to reduce variance?

To validate the first question, we check whether the policy gradient estimator remains unbiased, which requires proving that:

$$\mathbb{E}_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) b] = 0. \tag{2.34}$$

Expanding this expectation:

$$\mathbb{E}_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) b] = \int b \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) d\tau, \tag{2.35}$$

which simplifies to:

$$b \int \nabla_\theta \pi_\theta(\tau) d\tau = b \nabla_\theta 1 = 0. \tag{2.36}$$

Thus, the policy gradient remains an unbiased estimator even with a baseline.

To determine the optimal baseline b^*, we analyze the variance of the policy gradient:

$$\mathrm{Var}[\nabla_\theta J(\theta)] = \mathbb{E}_{\tau \sim p_\theta(\tau)}[(\nabla_\theta \log \pi_\theta(\tau)(R(\tau) - b))^2] - \mathbb{E}_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau)(R(\tau) - b)]^2. \tag{2.37}$$

Defining $g(\tau) = \nabla_\theta \log \pi_\theta(\tau)$ and differentiating with respect to b:

$$\frac{d}{db}\text{Var} = \frac{d}{db}(\mathbb{E}[g(\tau)^2 R(\tau)^2] - 2\mathbb{E}[g(\tau)^2 R(\tau) b] + b^2 \mathbb{E}[g(\tau)^2]). \tag{2.38}$$

Setting the derivative to zero yields:

$$b^* = \frac{\mathbb{E}[g(\tau)^2 R(\tau)]}{\mathbb{E}[g(\tau)^2]}. \tag{2.39}$$

Although computing this exact baseline is challenging in practical RL scenarios, a widely adopted approach is to define the baseline as the value function $V_\pi(s_t)$, which represents the expected return from state s_t under policy π:

$$b(s_t) = \mathbb{E}[r_t + \gamma r_{t+1} + ... + \gamma^T r_{t+T}] = V_\pi(s_t). \tag{2.40}$$

Using state values $V_\pi(s_t)$ ensures that action probabilities adjust based on deviations from expected returns, though learning this function introduces algorithmic complexity.

To further reduce variance in policy gradient estimation in the REINFORCE algorithm, we apply the principle of causality: rewards at time t influence only future rewards at $t' \geq t$. Thus, the policy gradient equation is rewritten as:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'}). \tag{2.41}$$

Here, the log-likelihood term $\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})$ is now multiplied by the return starting from time step t, rather than from step 0, as illustrated in

Fig. 3.4.

Policy Gradient Theorem

Sutton et al. [2] demonstrated that the policy gradient can be estimated using the reward-to-go $\sum_{t'=t}^{T-1} \gamma^{t'-t} r(s_{t'}, a_{t'}, s_{t'+1})$ instead of the Q-values:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\rho_\theta, a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q_{\pi_\theta}(s,a)], \tag{2.42}$$

where ρ_θ represents the distribution of states encountered under the policy π_θ. The REINFORCE algorithm can be considered a specific case of the policy gradient theorem, as the Q-value of an action approximates the reward-to-go after executing that action.

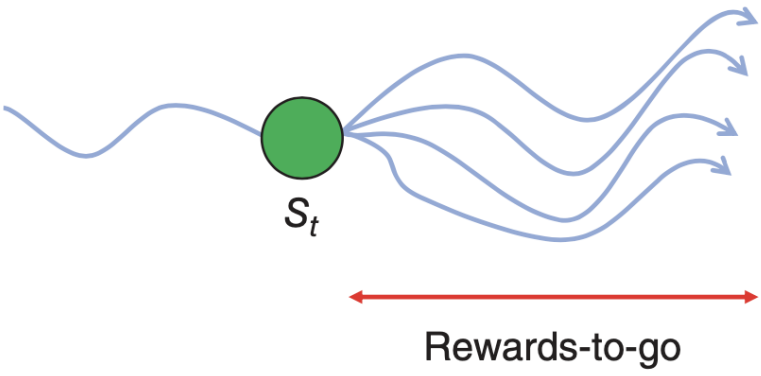


Figure 2.3: Reward-to-Go representation: summation of rewards collected starting from transition (s_t, a_t). Reproduced from [5].

Applying the policy gradient theorem, expectations are computed over individual transitions $\{(s_{i,t}, a_{i,t}, r_{i,t}, s_{i,t+1})\}$ instead of complete trajectories, enabling the use of bootstrapping techniques as in to Temporal Difference (TD) methods (Appendix A.2.3).

Since the true Q-values are typically unknown, they must be approximated. Sutton et al. [2] proved that Q-values can be estimated with a function approximator Q_ϕ, parameterized by ϕ, while still maintaining an unbiased estimate of the policy gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\rho_\theta, a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q_\phi(s,a)]. \tag{2.43}$$

The Q-value function must satisfy the Compatible Function Approximation Theorem, ensuring that the approximation aligns with the policy gradient condition:

$$\nabla_\phi Q_\phi(s,a) = \nabla_\theta \log \pi_\theta(s,a), \tag{2.44}$$

minimizing the mean squared error (MSE) between the true and approximated Q-values:

$$\mathbb{E}_{s\sim p_\theta, a\sim\pi_\theta}\left[(Q_{\pi_\theta}(s,a) - Q_\phi(s,a))^2\right]. \tag{2.45}$$

However, in DRL algorithms, these conditions are not always met.

Actor-Critic Methods

The introduction of Q-value approximators leads to a new class of DRL algorithms known as actor-critic methods. In these architectures, the actor network $\pi_\theta(s,a)$ learns the optimal policy, while the critic network estimates the Q-values $Q_\phi(s,a)$.

Most policy gradient DRL algorithms, such as A2C, TRPO and PPO, follow the actor-critic structure illustrated in Figure 3.5. Notably, actor and

critic networks can be updated using single transitions rather than entire trajectories. However, policy gradient methods inherently suffer from high variance. The following methods attempt to mitigate this variance and improve sample efficiency through techniques such as advantages, deterministic policies, and natural gradients. In actor-critic architectures, the actor and critic networks may be entirely separate or share certain parameters, such as convolutional layers in early stages.

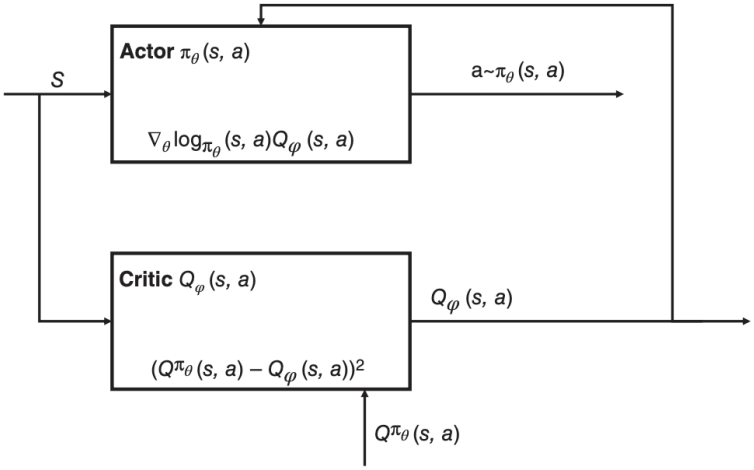


Figure 2.4: Actor-Critic architecture. The actor selects actions based on the current policy, while the critic evaluates the action by estimating the value function. The critic provides feedback to refine the policy iteratively. Reproduced from [5].

Advantage of Actor-Critic Methods

The policy gradient theorem underpins actor-critic methods, where the actor optimizes the policy while the critic estimates Q-values to refine the policy gradient. Unlike REINFORCE, which multiplies $\nabla_\theta \log \pi_\theta(a|s)$ by the return computed from a complete trajectory, actor-critic methods allow updates

based on single transitions. However, like REINFORCE, vanilla actor-critic methods suffer from high variance, necessitating a baseline to mitigate it. A natural choice is a state-dependent baseline, specifically the state-value function $V_\pi(s)$. The quantity multiplying the log-likelihood term in the policy gradient is:

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s), \tag{2.46}$$

where $A_\pi(s,a)$ represents the advantage of taking action a in state s. In Advantage Actor-Critic (A2C) methods, the advantage function is parameterized as A_ϕ:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\rho_\theta, a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A_\phi(s,a)], \tag{2.47}$$

where $A_\phi(s,a)$ approximates the expected advantage. Various techniques exist for estimating the advantage function:

- **Monte Carlo Advantage Estimate**: The Q-value $Q_{\pi_\theta}(s,a)$ is replaced with the actual return, yielding $A_\phi(s,a) = R(s,a) - V_\phi(s)$.
- **TD Estimate**: The advantage is computed as $A_\phi(s,a) = r(s,a,s') + \gamma V_\phi(s') - V_\phi(s)$.
- **n-Step Advantage Estimate**: The advantage function is estimated as:

$$A_\phi(s,a) = \sum_{k=0}^{n-1} \gamma^k r_{t+k+1} + \gamma^n V_\phi(s_{t+n+1}) - V_\phi(s_t). \tag{2.48}$$

Monte Carlo and Temporal Difference methods have inherent trade-offsMonte Carlo suffers from slow learning and dependency on finite tasks, while TD

methods may exhibit instability.

Advantage Actor-Critic (A2C)

Advantage Actor-Critic (A2C) utilizes the n-step advantage estimation method, blending elements of both TD and Monte Carlo approaches. While Monte Carlo estimates Q-values from the reward-to-go, TD computes expected returns using $V_\pi(s)$. The policy gradient for A2C is given by:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\rho_\theta, a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) \sum_{k=0}^{n-1} \gamma^k r_{t+k+1} + \gamma^n V_\phi(s_{t+n+1}) - V_\phi(s_t)\right]. \tag{2.49}$$

This technique strikes a balance between bias in TD learning and variance in Monte Carlo approaches, enhancing stability and efficiency. The A2C algorithm follows a structured process of sampling transitions, computing the n-step return, and updating policy parameters in an online manner. Further details such as the algorithmic formulation of A2C are available in Appendix C.

GAE

We recall the fundamental form of the policy gradient introduced at the beginning of our discussion on policy gradient techniques:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s_t\sim\rho_\theta, a_t\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(s_t,a_t)\psi_t], \tag{2.50}$$

where the term ψ_t can take different forms depending on the chosen method:

- $\psi_t = R_t$, for the REINFORCE algorithm (Monte Carlo sampling),

- $\psi_t = R_t - b$, for REINFORCE with baseline,
- $\psi_t = Q_\pi(s_t, a_t)$, for the policy gradient theorem,
- $\psi_t = A_\pi(s_t, a_t)$, for the advantage actor-critic method,
- $\psi_t = r_{t+1} + \gamma V_\pi(s_{t+1}) - V_\pi(s_t)$, for TD actor-critic,
- $\psi_t = \sum_{k=0}^{n-1} \gamma^k r_{t+k+1} + \gamma^n V_\pi(s_{t+n+1}) - V_\pi(s_t)$, for the n-step method (A2C).

In general, the term ψ_t, when multiplied by the gradient of the log-probabilities $\nabla_\theta \log \pi_\theta$, defines the expected return in the current state. The more ψ_t relies on actual rewards, the lower the bias but the higher the variance, necessitating a larger number of samples for accurate gradient estimation, increasing sample complexity. Conversely, when ψ_t is based on estimators such as TD error, the gradient is more stable (low variance) but may be inaccurate (high bias), leading to suboptimal policies. This is commonly referred to as the bias-variance trade-off in machine learning. The n-step method used in A2C provides a compromise between high variance and high bias.

GAE [6] was introduced to provide finer control over this bias-variance trade-off. The key idea is to combine advantages with the n-step method, leading to the following n-step advantage definition:

$$A_t^n = \sum_{k=0}^{n-1} \gamma^k r_{t+k+1} + \gamma^n V_\pi(s_{t+n+1}) - V_\pi(s_t), \tag{2.51}$$

which can also be rewritten in terms of TD errors:

$$A_t^n = \sum_{l=0}^{n-1} \gamma^l \delta_{t+l}, \tag{2.52}$$

where $\delta_t = r_{t+1} + \gamma V_\pi(s_{t+1}) - V_\pi(s_t)$ is the TD error.

Instead of selecting a fixed n-step value, GAE averages multiple n-step advantages while weighting them with a discount factor λ:

$$A_t^{GAE(\gamma,\lambda)} = (1-\lambda) \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}. \tag{2.53}$$

This formulation represents the discounted sum of all n-step advantages. If $\lambda = 0$, GAE reduces to the TD error method $A_t^{GAE(\gamma,0)} = \delta_t$ (high bias, low variance). Conversely, if $\lambda = 1$, GAE becomes equivalent to the Monte Carlo advantage $A_t^{GAE(\gamma,1)} = R_t$, leading to low bias but high variance. Thus, tuning λ between 0 and 1 enables a flexible balance between bias and variance.

It is important to note that γ and λ serve distinct roles in policy gradient estimation. The discount factor γ determines the importance of future rewards, with high values ($\gamma < 1$) leading to lower bias but higher variance. Schulman et al. [6] empirically demonstrated that setting λ slightly lower than γ (typically around 0.99) results in lower bias.

The final expression for the policy gradient incorporating GAE is:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s_t \sim \rho_\pi, a_t \sim \pi_\theta} \left[\nabla_\theta \log \pi_\theta(s_t, a_t) \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l} \right]. \tag{2.54}$$

We summarize the GAE method in Algorithm 1.

Algorithm 1 GAE Pseudocode

Input: n-step size n, episode length T, learning rate η
Initialize actor π_θ and critic V_ϕ with random weights
Observe initial state s_0
for $t = 1, \ldots, T$ **do**
    Initialize empty minibatch
    for $k = 0, \ldots, n$ **do**
        Select action $a_k \sim \pi_\theta(s_k)$ and receive next state s_{k+1} and reward r_k
        Store (s_k, a_k, r_k, s_{k+1}) in minibatch
    end for
    for $k = 0, \ldots, n$ **do**
        Compute TD error: $\delta_k = r_k + \gamma V_\phi(s_{k+1}) - V_\phi(s_k)$
    end for
    for $k = 0, \ldots, n$ **do**
        Compute GAE advantage: $A_k^{GAE(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{k+l}$
    end for
    Update actor using GAE advantage and natural gradients (TRPO)
    Update critic using natural gradients (TRPO)
end for

2.5.3 Natural Gradients

In DRL, neural networks are commonly used as function approximators and are optimized using gradient-based methods such as Stochastic Gradient Descent (SGD) and its improved variants like RMSProp and Adam. Standard SGD updates the model parameters θ by moving in the direction of the negative gradient of the loss function (gradient descent) or in the positive direction for policy gradient (gradient ascent), scaled by a learning rate η:

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(\theta). \tag{2.55}$$

This approach, often referred to as the steepest descent method, seeks the smallest parameter changes that yield the largest improvement in the ob-

jective function. While this works well in supervised learning by stabilizing weight updates, it presents challenges in DRL where nonstationarity is a major issue.

One key difficulty in DRL arises from the shifting target values used to train the critic or Q-function. For instance, in Q-learning, the target value $r(s, a, s') + \gamma \max_{a'} Q_\theta(s', a')$ evolves along with the parameters θ. If Q-values fluctuate significantly across minibatches, the network's target values become unstable, leading to suboptimal policies. To address this, target networks are often employed, either by keeping a delayed version of the trained network, as in DQN (Appendix B). Though this introduces bias, the effect diminishes over time at the cost of increased sample complexity.

For on-policy methods, target networks cannot be employed, as the critic must always be trained on transitions generated by the latest version of the actor. This results in inefficiency, as older transitions become obsolete and cannot contribute to network updates. The policy gradient theorem highlights this issue:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\rho_\theta, a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) Q_{\pi_\theta}(s, a)]. \tag{2.56}$$

If the policy π_θ undergoes rapid changes, the estimated Q-values $Q_\phi(s, a)$ may no longer correspond to the actual policy, leading to bias and instability. Consequently, the actor should not change too quickly relative to the critic. A naive solution is to use a slow learning rate for the critic, but this severely increases the training time and sample complexity.

To overcome this issue, it is necessary to update parameters in a manner

that maximizes policy improvement while keeping policy changes minimal. The goal is to achieve a significant parameter shift that still preserves the usability of past experiences, thereby stabilizing learning.

Natural gradient methods address this problem by leveraging information geometry to optimize over probability distributions rather than raw parameters. Initially introduced in neural network training [7], natural gradients were later applied to policy gradient methods [8] and further extended into the natural actor-critic algorithm [9]. The core principle of natural gradients forms the basis of Trust Region Policy Optimization (TRPO) [10] and PPO [11], which have demonstrated superior performance in reinforcement learning tasks with continuous action spaces due to their improved sample efficiency and robustness to hyperparameters.

Principle of Natural Gradients

Consider two Gaussian distributions: $\mathcal{N}(0, 0.2)$ and $\mathcal{N}(1, 0.2)$. Another pair of distributions is $\mathcal{N}(0, 10)$ and $\mathcal{N}(1, 10)$. Although the Euclidean distance between their parameters is identical, the first pair represents significantly different distributions, whereas the second pair remains closely related. This illustrates that Euclidean distance is not an ideal measure of similarity between probability distributions, see Figure 2.5.

A more appropriate metric is the KullbackLeibler (KL) divergence:

$$D_{KL}(p||q) = \mathbb{E}_{x\sim p}\left[\log \frac{p(x)}{q(x)}\right] = \int p(x) \log \frac{p(x)}{q(x)} dx. \tag{2.57}$$

This divergence measures the statistical difference between distributions p

and q, reaching zero only when they are identical. However, KL divergence is asymmetric, i.e., $D_{KL}(p||q) \neq D_{KL}(q||p)$. A symmetrized version is the JensenShannon (JS) divergence:

$$D_{JS}(p||q) = \frac{1}{2}(D_{KL}(p||q) + D_{KL}(q||p)). \tag{2.58}$$

Euclidean distance does not reflect the manifold structure of probability distributions, a more suitable Riemannian metric is needed. The Fisher Information Matrix (FIM), derived from the second-order expansion of the KL divergence, provides this metric:

$$F(\theta) = \mathbb{E}_{x\sim p(x;\theta)} \left[\nabla_\theta \log p(x;\theta) \nabla_\theta \log p(x;\theta)^T\right]. \tag{2.59}$$

In policy optimization, a parameterized probability distribution $p(x;\theta)$ is modified to $p(x;\theta + \Delta\theta)$. Since Using the FIM, we approximate the KL divergence between two distributions:

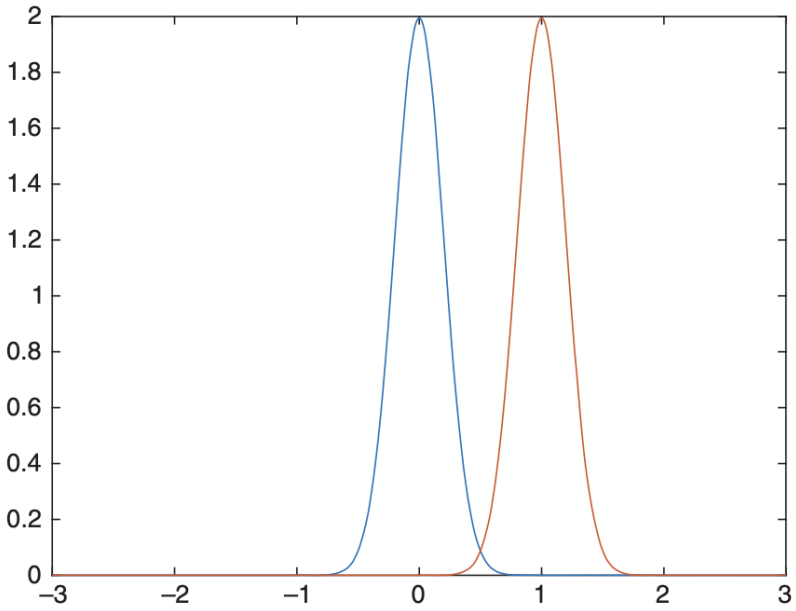
$$D_{KL}(p(x;\theta)||p(x;\theta + \Delta\theta)) \approx \Delta\theta^T F(\theta) \Delta\theta. \tag{2.60}$$

Natural gradient descent updates parameters using the inverse of the Fisher matrix:

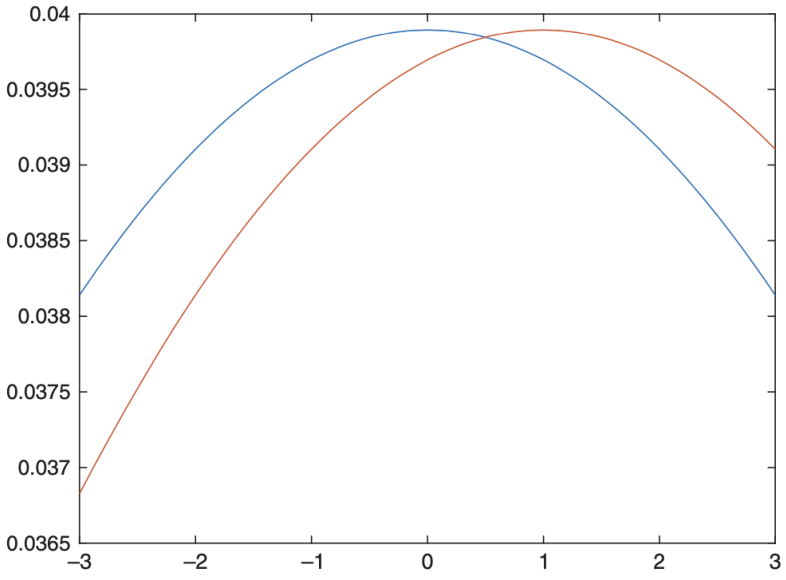
$$\nabla_\theta \mathcal{L}(\theta) = F(\theta)^{-1} \nabla_\theta \mathcal{L}(\theta). \tag{2.61}$$

Thus, the update rule follows:

$$\theta_{k+1} = \theta_k - \eta \nabla_\theta \mathcal{L}(\theta). \tag{2.62}$$



(a) Gaussian distributions $\mathcal{N}(0, 0.2)$ and $\mathcal{N}(1, 0.2)$.



(b) Gaussian distributions $\mathcal{N}(0, 10)$ and $\mathcal{N}(1, 10)$.

Figure 2.5: Comparison between two pairs of Gaussian distributions. The Euclidean distance in the parameter space $d = \sqrt{(\mu_1 - \mu_2)^2 + (\sigma_1 - \sigma_2)^2}$ is the same for both pairs. However, the distributions in the first pair (Figure 2.5a) are clearly more separated, while in the second pair (Figure 2.5b), they are much closer.

In flat regions, natural gradients take larger steps, while in steep regions, steps are smaller, preventing instability.

Natural gradient descent has proven effective in optimizing deep networks for supervised learning [12], though it requires computing the inverse of the Fisher matrix, which is computationally expensive. Schulman et al. [10] incorporated natural gradients into policy optimization with TRPO, addressing computational challenges and making reinforcement learning more efficient.

TRPO

TRPO was introduced by Schulman et al. [10] as a technique to incorporate natural gradients into policy optimization, leveraging nonlinear function approximators such as neural networks. The primary goal in DRL is to optimize the expected return, defined as:

$$\eta(\pi) = \mathbb{E}_{s\sim\rho_\pi, a\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t, a_t, s_{t+1})\right], \tag{2.63}$$

where ρ_π represents the state distribution induced by policy π:

$$\rho_\pi(s) = P(s_0 = s) + \gamma P(s_1 = s) + \gamma^2 P(s_2 = s) + \dots. \tag{2.64}$$

The improvement of a policy π_θ over an older policy $\pi_{\theta_{old}}$ can be expressed in terms of the advantage function [13]:

$$\eta(\theta) = \eta(\theta_{old}) + \mathbb{E}_{s\sim\rho_{\pi_\theta}, a\sim\pi_\theta}[A_{\pi_{\theta_{old}}}(s, a)]. \tag{2.65}$$

Since computing this expectation is infeasible due to the dependency on ρ_{π_θ},

we approximate it by assuming that the new policy remains close to the old policy, allowing us to sample from $\rho_{\pi_{\theta_{old}}}$:

$$\eta(\theta) \approx \eta(\theta_{old}) + \mathbb{E}_{s\sim\rho_{\pi_{\theta_{old}}},a\sim\pi_\theta}[A_{\pi_{\theta_{old}}}(s,a)]. \tag{2.66}$$

TRPO ensures stable policy updates by applying constraints on policy divergence. The surrogate objective function is defined as:

$$\max_\theta J_{\theta_{old}}(\theta) = \eta(\theta_{old}) + \mathbb{E}_{s\sim\rho_{\pi_{\theta_{old}}},a\sim\pi_\theta}[A_{\pi_{\theta_{old}}}(s,a)], \quad \text{subject to } D_{KL}(\pi_{\theta_{old}}||\pi_\theta) \leq \delta. \tag{2.67}$$

This ensures that the updated policy remains within a trust region of the previous policy, preventing large, unstable updates. An alternative formulation introduces a soft constraint by incorporating a penalty term in the objective function:

$$\max_\theta \mathcal{L}(\theta) = J_{\theta_{old}}(\theta) - CD_{KL}(\pi_{\theta_{old}}||\pi_\theta), \tag{2.68}$$

where C is a hyperparameter controlling the trade-off between policy improvement and constraint satisfaction.

Since computing the maximum KL divergence across the entire state space is intractable, it is approximated using the expected KL divergence:

$$D_{KL}(\pi_{\theta_{old}}||\pi_\theta) \approx \mathbb{E}_{s\sim\rho_{\pi_{\theta_{old}}}}\left[D_{KL}(\pi_{\theta_{old}}(s,\cdot)||\pi_\theta(s,\cdot))\right]. \tag{2.69}$$

Optimization via Importance Sampling

Although Schulman et al. [10] provided a theoretical foundation using regularized optimization, the practical implementation adopts a constrained optimization approach:

$$\max_{\theta} J_{\theta_{old}}(\theta) = \eta(\theta_{old}) + \mathbb{E}_{s\sim\rho_{\pi_{\theta_{old}}}, a\sim\pi_{\theta}}[A_{\pi_{\theta_{old}}}(s,a)], \quad \text{subject to } D_{KL}(\pi_{\theta_{old}}||\pi_{\theta}) \leq \delta. \tag{2.70}$$

Since $\eta(\theta_{old})$ does not depend on θ, it is treated as a constant in this optimization. The goal is to maximize the advantage function over actions selected by the new policy π_θ in states sampled from the old policy $\pi_{\theta_{old}}$. However, direct sampling from π_θ is not feasible, so importance sampling is introduced to utilize samples from the old policy:

$$\max_{\theta} \mathbb{E}_{s\sim\rho_{\pi_{\theta_{old}}}, a\sim\pi_{\theta_{old}}} \left[\frac{\pi_\theta(s,a)}{\pi_{\theta_{old}}(s,a)} A_{\pi_{\theta_{old}}}(s,a) \right], \quad \text{subject to } D_{KL}(\pi_{\theta_{old}}||\pi_{\theta}) \leq \delta. \tag{2.71}$$

This allows for unbiased estimation of the policy gradient while constraining divergence from the old policy.

In some cases, the advantage function can be replaced with Q-values to simplify optimization. Since advantages are defined as:

$$A_{\pi_{\theta_{old}}}(s,a) = Q_{\pi_{\theta_{old}}}(s,a) - V_{\pi_{\theta_{old}}}(s), \tag{2.72}$$

where the state value function $V_{\pi_{\theta_{old}}}(s)$ does not depend on the new policy,

it can be omitted in optimization:

$$\max_{\theta} \mathbb{E}_{s\sim\rho_{\pi_{\theta_{old}}}, a\sim\pi_{\theta_{old}}} \left[\frac{\pi_\theta(s,a)}{\pi_{\theta_{old}}(s,a)} Q_{\pi_{\theta_{old}}}(s,a) \right]. \tag{2.73}$$

TRPO iteratively solves this constrained optimization problem using second-order optimization techniques to ensure stable policy updates.

Practical Implementation

To solve the constrained optimization problem in TRPO, the Lagrange method is employed with a multiplier λ:

$$\mathcal{L}(\theta, \lambda) = J_{\theta_{old}}(\theta) - \lambda \left(D_{KL}(\pi_{\theta_{old}} || \pi_\theta) - \delta \right). \tag{2.74}$$

This formulation is closely related to the regularized approach discussed in Equation (3.81). To further simplify the optimization, a second-order approximation of the KL divergence is applied using the Fisher Information Matrix:

$$D_{KL}(\pi_{\theta_{old}} || \pi_\theta) \approx (\theta - \theta_{old})^T F(\theta_{old})(\theta - \theta_{old}). \tag{2.75}$$

Substituting this into the Lagrangian function, we obtain:

$$\mathcal{L}(\theta, \lambda) = \nabla_\theta J_{\theta_{old}}(\theta - \theta_{old}) - \lambda(\theta - \theta_{old})^T F(\theta_{old})(\theta - \theta_{old}). \tag{2.76}$$

This function is quadratic in $\Delta\theta = \theta - \theta_{old}$, meaning that its maximum is achieved when the first derivative is zero:

$$\nabla_\theta J_{\theta_{old}}(\theta) = \lambda F(\theta_{old})\Delta\theta. \tag{2.77}$$

Solving for $\Delta\theta$ gives:

$$\Delta\theta = \frac{1}{\lambda} F(\theta_{old})^{-1} \nabla_\theta J_{\theta_{old}}(\theta). \tag{2.78}$$

This expression describes the update step in natural gradient descent. The step size $1/\lambda$ must be determined, though it can be replaced with a fixed hyperparameter.

Computing the inverse of the Fisher Information Matrix is computationally expensive, as it scales quadratically with the number of parameters. In the original paper [10], the authors propose using a conjugate gradient algorithm followed by a line search to find the next policy parameters θ_{k+1} while ensuring that the KL divergence constraint is satisfied.

Summary of TRPO Characteristics TRPO exhibits several key properties:

- It is a policy gradient method that improves policy performance monotonically.
- It employs a surrogate objective function that lower-bounds the expected return, ensuring stable updates while preventing drastic policy changes via KL divergence constraints.

- It is less sensitive to the learning rate compared to other methods.

However, TRPO also presents some challenges:

- It is difficult to apply to neural networks with multiple outputs, such as those used for both policy and value function estimation, as it relies on policy distribution constraints.
- While TRPO performs well with fully connected layers, it is less efficient when applied to convolutional neural networks (CNNs) or recurrent neural networks (RNNs).
- The conjugate gradient implementation is more complex than standard stochastic gradient descent (SGD).

2.6 PPO

2.6.1 Introduction to PPO

The evolution of policy gradient methods has been driven by the need to reduce variance, improve sample efficiency, and stabilize policy updates. Below is a structured recap leading to TRPO:

1. **Policy Gradient Theorem**

   The policy gradient theorem (2.42) provides a direct method for optimizing policies but suffers from high variance, requiring a large number of trajectory samples for stable learning.

2. **Variance Reduction: Baselines and Advantage Estimation**

   Introducing a baseline, typically the state-value function $V_\pi(s)$, leads

to the advantage function (2.46), improving sample efficiency while maintaining unbiased policy gradient estimates.

3. **GAE**

   GAE (2.53) further stabilizes policy updates by balancing bias and variance using a weighted sum of temporal differences.

4. **Natural Gradients and Trust Regions**

   To ensure stable updates, natural gradients (2.59) account for the curvature of the parameter space, but their computational cost is high.

5. **TRPO**

   TRPO enforces a KL divergence constraint between successive policies (2.71) to ensure gradual updates. While effective, it requires solving a constrained optimization problem using second-order methods, increasing computational complexity.

PPO was introduced to address the computational inefficiencies of TRPO, which requires solving a constrained optimization problem with second-order methods. PPO simplifies policy updates by replacing the KL divergence constraint with a clipped objective function, reducing computational overhead while preserving stability.

As a natural policy gradient method, PPO belongs to the family of policy gradient approaches, optimizing policies by leveraging the geometry of the probability distribution space. Unlike TRPO, which imposes a strict constraint on policy updates, PPO introduces a clipping mechanism that prevents excessively large updates, ensuring smoother and more stable learn-

ing.

The core advantage of PPO lies in its ability to balance exploration and exploitation while maintaining robust training dynamics. By limiting drastic policy changes, PPO improves sample efficiency, making it particularly effective for optimizing policies in complex environments, such as those involving continuous action spaces.

Additionally, PPO enhances the applicability of policy gradient methods to diverse neural network architectures, including convolutional neural networks (CNNs) and recurrent neural networks (RNNs). By modifying the surrogate objective function into a more tractable expression, PPO allows optimization through first-order gradient ascent techniques, improving its usability across a broader range of reinforcement learning tasks.

2.6.2 Clipped Surrogate Objective

The primary idea behind PPO is to reformulate the surrogate objective function, which acts as a lower bound to the expected return, into a more tractable expression that can be efficiently optimized using first-order techniques such as gradient ascent.

To achieve this, the surrogate objective function of TRPO is rewritten by incorporating the time index and the sampling weight ρ_t:

$$\mathcal{L}_{\mathrm{CPI}}(\theta) = \mathbb{E}_t \left[\frac{\pi_\theta(s_t, a_t)}{\pi_{\theta_{\mathrm{old}}}(s_t, a_t)} A^{\pi_{\theta_{\mathrm{old}}}}(s_t, a_t) \right] = \mathbb{E}_t \left[\rho_t(\theta) A^{\pi_{\theta_{\mathrm{old}}}}(s_t, a_t) \right]. \quad (2.79)$$

The term "CPI" refers to Conservative Policy Iteration [13]. Without

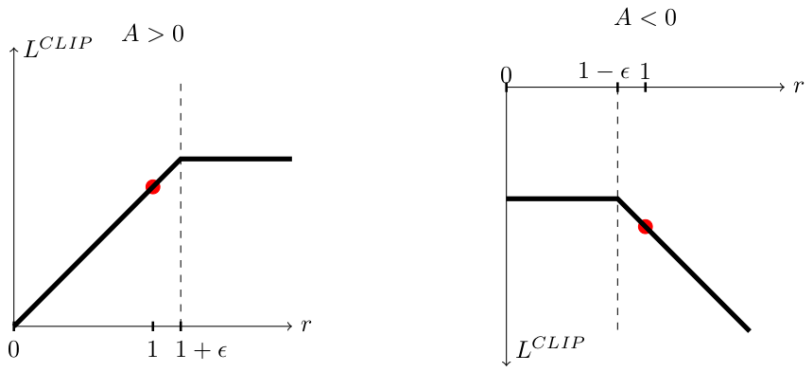


Figure 2.6: Clipping effect in PPO. (a) $A > 0$, (b) $A < 0$.

the KL constraint, policy updates may become excessively large, leading to instability. To address this, PPO introduces a mechanism to penalize policy changes that result in significant deviations of the sampling weight ρ_t from 1. This effectively controls policy shifts where the KL divergence is large.

To achieve this constraint implicitly, the authors propose the following clipped objective function:

$$\mathcal{L}_{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(\rho_t(\theta) A^{\pi_{\theta_{\text{old}}}}(s_t, a_t), \text{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon) A^{\pi_{\theta_{\text{old}}}}(s_t, a_t) \right) \right]. \tag{2.80}$$

The `min` function ensures that the left term remains equivalent to the surrogate function used in TRPO, while the right term restricts the importance sampling weight to stay close to 1 by applying a clipping parameter ϵ. This modification prevents overly aggressive policy updates, enhancing stability and improving sample efficiency. This objective ensures that the policy update remains stable by clipping the sampling weight r_t within a defined range $[1-\epsilon, 1+\epsilon]$. When $A(s_t, a_t) > 0$, the policy can increase the

probability of selecting an advantageous action but is constrained to prevent excessive growth. Similarly, for $A(s_t, a_t) < 0$, the probability of a less advantageous action decreases, but large reductions are avoided. As illustrated in Figure 2.6, when the advantage $A(s_t, a_t) > 0$, i.e., selecting action a_t leads to a higher return than the expected action, the selection probability $\pi_\theta(s_t, a_t)$ increases. However, if ρ_t becomes excessively large, it is clipped at $1 + \epsilon$, ensuring stability. Similarly, for $A(s_t, a_t) < 0$, the selection probability decreases, and ρ_t is clipped at $1 - \epsilon$ to avoid drastic reductions.

2.7 Exploration and Exploitation in PPO

PPO trains a stochastic policy in an on-policy fashion, where the same policy used to collect trajectories is updated during training. This design inherently supports exploration by sampling actions according to the probability distribution produced by the latest version of its policy, $\pi_\theta(a|s)$.

At the start of training, the policy is initialized with a high degree of randomness, enabling extensive exploration of the state-action space. This stochastic behavior helps the agent identify high-reward regions of the environment that might otherwise be overlooked. As the training progresses, the policy gradually becomes less random. This transition occurs because the update rule in PPO prioritizes actions with higher advantage estimates $A(s, a)$, steering the policy towards more deterministic behavior that exploits known rewards.

However, this shift from exploration to exploitation introduces the risk of getting trapped in local optima. As the policy becomes more deterministic,

it may cease exploring alternative strategies, particularly in environments with complex or deceptive reward landscapes. This premature convergence can result in suboptimal policies if the agent fails to adequately explore all possible actions. To address this issue, PPO incorporates an entropy bonus in its loss function.

In reinforcement learning, the concept of an entropy bonus is introduced to encourage exploration during training. Entropy measures the randomness or unpredictability of a probability distribution. Formally, for a random variable x with probability mass or density function P, entropy $H(P)$ is defined as:

$$H(P) = \mathbb{E}_{x\sim P}[-\log P(x)]. \tag{2.81}$$

In entropy-regularized reinforcement learning, the agent receives an additional reward at each timestep proportional to the entropy of its policy. This modifies the standard reinforcement learning problem to include an entropy bonus, leading to the following optimization objective:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t \Big(R_t + \alpha H\left(\pi(\cdot|s_t)\right) \Big)\right], \tag{2.82}$$

where $\alpha > 0$ is a trade-off coefficient that balances the standard reward R and the entropy bonus. The parameter α determines how much exploration is incentivized compared to exploitation.

The inclusion of the entropy bonus modifies the definitions of the value functions. The state value function $V^\pi(s)$ now includes entropy bonuses from

all timesteps:

$$V^{\pi}(s) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^{t}\Big(R_t + \alpha H\left(\pi(\cdot|s_t)\right)\Big)\Big|s_0 = s\right]. \tag{2.83}$$

Similarly, the state-action value function $Q^{\pi}(s,a)$ incorporates entropy bonuses starting from the second timestep:

$$Q^{\pi}(s,a) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^{t}R_t + \alpha\sum_{t=1}^{\infty}\gamma^{t}H\left(\pi(\cdot|s_t)\right)\Big|s_0 = s, a_0 = a\right]. \tag{2.84}$$

These modified value functions are connected through the following relationship:

$$V^{\pi}(s) = \mathbb{E}_{a\sim\pi}\left[Q^{\pi}(s,a)\right] + \alpha H\left(\pi(\cdot|s)\right). \tag{2.85}$$

The entropy bonus encourages policies to maintain randomness during training, preventing premature convergence to deterministic behaviors. This approach is particularly beneficial in stochastic environments or when the agent requires extensive exploration to discover optimal strategies. The trade-off coefficient α plays a critical role in balancing exploration and exploitation.

While the definitions of value functions in entropy-regularized reinforcement learning can vary slightly across literature, the underlying principle remains consistent: incorporating entropy into the learning process promotes a robust and exploratory policy, enabling the agent to better handle complex and dynamic environments.

To conclude, in algoritmh 2 we propose an implementation of the pseudocode for the algorithm, illustrating its workflow and main components,

while in algorithm 3 we propose the variant with entropy bonus exploration.

Algorithm 2 PPO-Clip

Input: initial policy parameters θ_0, initial value function parameters ϕ_0
for $k = 0, 1, 2, \ldots$ **do**
    Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
    Compute rewards-to-go $\hat{R}_t$.
    Compute advantage estimates $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
    Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} \hat{A}^{\pi_{\theta_k}}(s_t, a_t),\ g(\epsilon, \hat{A}^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

    typically via stochastic gradient ascent with Adam.
    Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

    typically via some gradient descent algorithm.
end for

Algorithm 3 PPO-Clip with Entropy Bonus

1: **Input:** initial policy parameters θ_0, initial value function parameters ϕ_0, entropy coefficient β

2: **for** $k = 0, 1, 2, \ldots$ **do**

3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.

4: Compute rewards-to-go $\hat{R}_t$.

5: Compute advantage estimates $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.

6: Update the policy by maximizing the PPO-Clip objective with entropy bonus:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left[\min \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} \hat{A}^{\pi_{\theta_k}}(s_t, a_t),\ g(\epsilon, \hat{A}^{\pi_{\theta_k}}(s_t, a_t)) \right) + \beta H(\pi_\theta(\cdot|s_t)) \right]$$

where $H(\pi_\theta(\cdot|s_t)) = -\sum_{a \in \mathcal{A}} \pi_\theta(a|s_t) \log \pi_\theta(a|s_t)$ represents the entropy of the policy.

7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

8: **end for**

Chapter 3

Antenna Selection by Reinforcement Learning

3.1 Introduction

This chapter builds upon the theoretical foundations introduced in the previous chapters, where the principles of Wi-Fi communication and reinforcement learning were discussed. Here, we formalize the antenna selection problem within a reinforcement learning framework and present a deep learning-based approach to dynamically optimize antenna configurations.

Antenna selection is a crucial aspect of modern wireless communication systems, influencing spectral efficiency, interference mitigation, and power consumption. The increasing demand for high-speed, reliable wireless connections requires adaptive selection strategies capable of responding to real-time variations in the communication environment. Traditional approaches rely on static heuristics or predefined selection rules, which often struggle

to generalize across different network conditions. Reinforcement learning, on the other hand, provides a data-driven alternative that allows an agent to autonomously adapt its behavior through continuous interaction with the environment.

To address this problem, we model the antenna selection process as a MDP, defining a state space that includes real-time channel measurements and other key performance indicators. The proposed solution employs PPO. To enhance temporal modeling capabilities, the agent utilizes a recurrent neural network structure based on Gated Recurrent Unit (GRU), allowing it to capture historical dependencies in channel state variations. This enables the model to make more informed and predictive decisions, rather than relying solely on instantaneous observations.

The chapter is structured as follows. First, we describe the system model (3.2), outlining the network setup and the key performance indicators used to assess channel conditions. Next, we present the mathematical formulation (3.3) of the antenna selection problem as an MDP, specifying the state representation, action space, and reward function. Following this, we introduce the deep reinforcement learning framework (3.4), detailing the neural network architecture, policy optimization methodology, and training process. Finally, we evaluate the proposed approach in Chapter 4 - Performance Evaluation through a series of experiments in both static and dynamic environments, comparing its performance against a baseline antenna selection policy. This evaluation assesses the effectiveness of reinforcement learning in optimizing antenna selection under varying wireless conditions.

The thesis then concludes in Chapter 5 - Conclusions, where we discuss

the results, highlighting the advantages of reinforcement learning in wireless optimization and outlining potential directions for future research.

Through this structured approach, we demonstrate the effectiveness of reinforcement learning in antenna selection, showcasing its ability to adapt dynamically to changing wireless conditions and optimize network performance.

3.2 System Model

We consider a commercial WiFi router operating under the IEEE 802.11ax standard, commonly referred to as WiFi 6, and configured to use the 5 GHz band for domestic applications. The AP leverages advanced MIMO technologies and is equipped with L RF chains, each connected to N_T antennas. While the client is equiped with N_u antenna each one connected to its RF chain. The AP can dynamically activate only a subset of L among available antennas at any given time, leading to a total of $N = {N_T}^L$ possible combinations, as shown in Figure 3.1.

The system operates in a time-division duplex (TDD) mode, leveraging channel reciprocity to ensure identical uplink and downlink channels. Let $\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_{N_T}]^T$ denote the $N_T \times N_u$ full CSI (Channel State Information) matrix, where $\mathbf{h}_j$ represents the channel vector for the j-th receive antenna. The received signal at the access point can be expressed as:

$$\mathbf{y} = \sqrt{\rho}\mathbf{H}\mathbf{x} + \mathbf{w}, \tag{3.1}$$

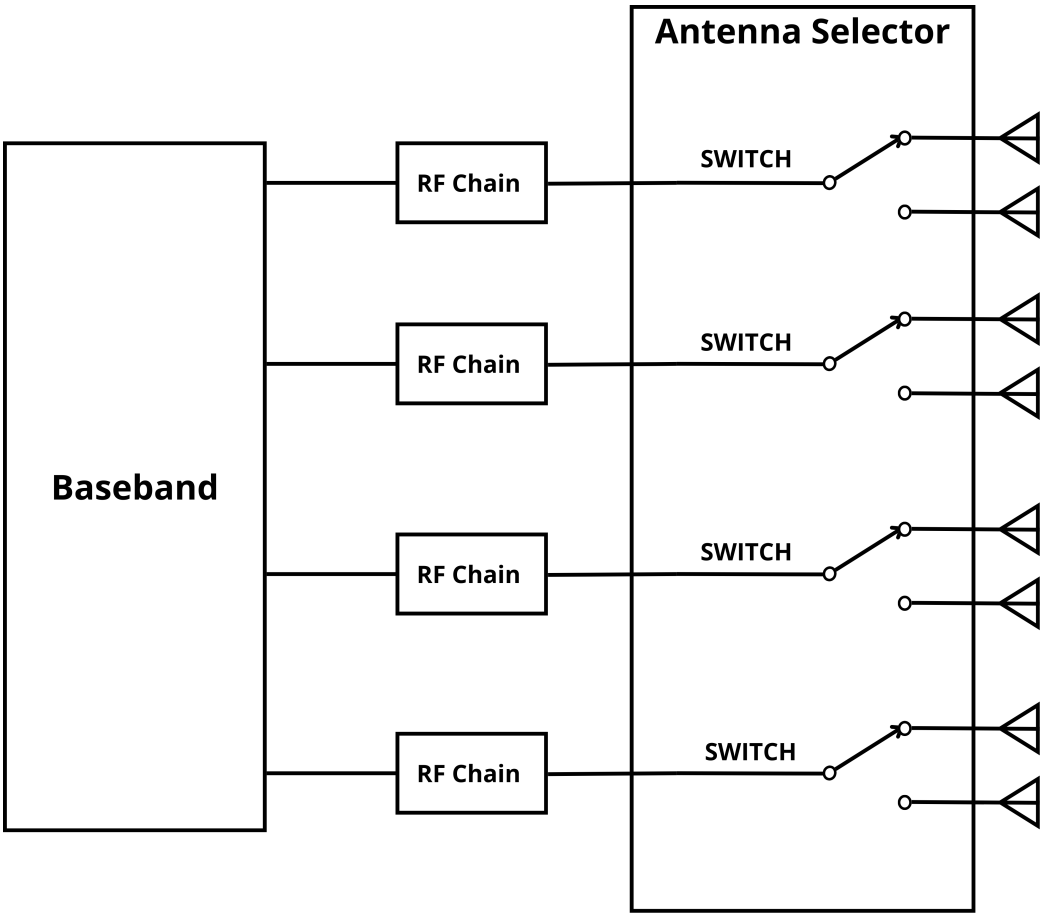


Figure 3.1: AP radio scheme

where $\mathbf{x} \in \mathbb{C}^{N_u \times 1}$ is the transmit symbol vector with unit power, $\mathbf{w}$ is the additive white Gaussian noise (AWGN) vector with zero mean and unit variance, and ρ denotes the average signal-to-noise ratio (SNR).

The capacity of the system is given by:

$$C(\mathbf{H}) = B \log_2 \det(\mathbf{I}_{N_u} + \rho \mathbf{H}^H \mathbf{H}), [\frac{\text{bit}}{\text{s}}] \tag{3.2}$$

where $\mathbf{I}_{N_u}$ is the $N_u \times N_u$ identity matrix, $\det(\cdot)$ represents the determinant, and $\mathbf{H}^H$ denotes the Hermitian transpose of $\mathbf{H}$ and B is the bandwidth. Let $\mathcal{A}$

represent the set of all possible antenna combinations, with $|\mathcal{A}|= {N_T}^L$. For a specific combination $\mathbf{a} \in \mathcal{A}$, the corresponding channel sub-matrix is denoted as $\mathbf{H}(\mathbf{a})$. Suppose that $|\mathcal{A}|= M$, at any given instant, there are M possible antenna configurations $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_M$, and correspondingly, M potential channel realizations, one for each configuration: $\mathbf{H}(\mathbf{a}_1), \mathbf{H}(\mathbf{a}_2), \ldots, \mathbf{H}(\mathbf{a}_M)$. The (3.3) becomes :

$$C(\mathbf{H}(\mathbf{a})) = \mathbf{B} \log_2 \det(\mathbf{I}_{N_u} + \rho \mathbf{H}(\mathbf{a})^H \mathbf{H}(\mathbf{a})), [\frac{\text{bit}}{\text{s}}] \tag{3.3}$$

Where we have now introduced $\mathbf{H}(\mathbf{a})$ into the capacity formula. Naturally, only the CSI of the corresponding L selected antennas at the current time slot can be measured.

The optimal selection maximizes the capacity:

$$\mathbf{a}^* = \arg \max_{\mathbf{a} \in \mathcal{A}} (C(\mathbf{H}(\mathbf{a})) \tag{3.4}$$

The capacity in (3.4) considers a static scenario with a single optimal antenna set $\mathbf{a}^*$. In reality, however, channel conditions evolve over time due to interference, mobility, and environmental changes, rendering the channel matrix $\mathbf{H}$ time-varying. Since the channel remains approximately constant only for a coherence time T_c, after each interval the channel realization changes. As a result, the antenna selection must be periodically updated to maintain or maximize communication throughput. Suppose we divide the transmission time between the AP and the client into time blocks, each of duration T_c. Let the total number of these time blocks be T, indexed by $j = 0, 1, 2, \ldots, T$. Within each time block j, we assume the channel remains approximately

constant. Therefore, for each time block j, we can solve the optimization problem in (3.4) to find the optimal antenna combination $\mathbf{a}^*(j)$:

$$\mathbf{a}^*(\mathbf{j}) = \arg\max_{\mathbf{a}\in\mathcal{A}}(C(\mathbf{H}(\mathbf{a},\mathbf{j})) \tag{3.5}$$

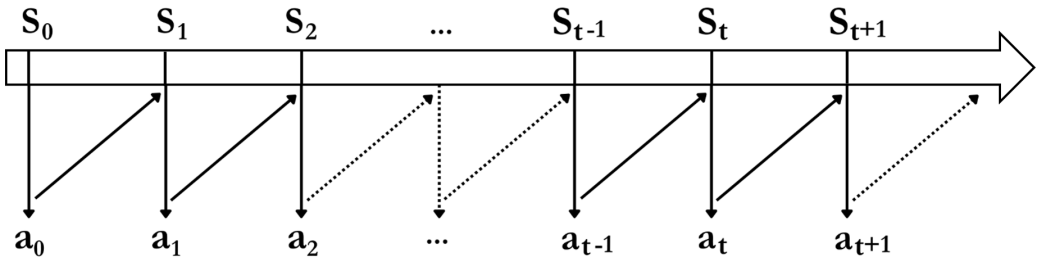


Figure 3.2: Illustration of a antenna selection process

By performing this optimization at each time block, we obtain a sequence of optimal antenna combinations over time:

$$\boldsymbol{\pi}^* = \{\mathbf{a}_0^*,\ \mathbf{a}_1^*,\ \mathbf{a}_2^*,\ \ldots,\ \mathbf{a}_T^*\}\,. \tag{3.6}$$

While the above formulation provides a sequence of optimal choices for a finite horizon T, in practical scenarios the communication process can be considered ongoing, and the goal is to maximize the expected capacity over an indefinitely long period. Let $\boldsymbol{\pi}$ denote a policy that specifies which N antennas to select at every time slot. If $\pi(j)$ denotes the antenna subset selected at time block j under policy π, and $C(\mathbf{H}(j,\pi(j)))$ is the instantaneous capacity at time slot j, we seek to determine a policy $\boldsymbol{\pi}$ that maximizes the expected long-term transmission rate. Formally, this can be expressed as:

$$R(\pi) = \lim_{t\to\infty}\frac{1}{t}\mathbb{E}\left[\sum_{j=0}^{t} C(\mathbf{H}(j,\pi(j)))\right]. \tag{3.7}$$

Here, the expectation $\mathbb{E}[\cdot]$ is taken over the stochastic evolution of the channel $\mathbf{H}$ and any randomness in the policy π. The objective is to find the policy π^* that achieves the supremum of this limit, thereby maximizing the expected long-term capacity.

3.2.1 Key Performance Indicators (KPIs) for Channel Estimation on the AP

In practical scenarios, obtaining complete CSI $\mathbf{H}(t)$ at every time instant is often infeasible. Instead, the system relies on various KPIs provided by the network interface hardware to infer channel conditions. The primary KPIs utilized are defined as follows:

- **CFR:** The CFR represents the frequency-domain response of the wireless channel and is expressed as a matrix:

$$\mathbf{CFR} = \underbrace{\left.\begin{bmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,n} \\ h_{2,1} & h_{2,2} & \cdots & h_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ h_{L,1} & h_{L,2} & \cdots & h_{L,n} \end{bmatrix}\right\} L \text{ transmitting antennas}}_{n \text{ subcarriers}} \tag{3.8}$$

  where $h_{i,k}$ is the channel gain for the i-th transmitting antenna and the k-th subcarrier. Each element $h_{i,k}$ is a complex value that captures the

magnitude and phase of the channel at the corresponding frequency and spatial stream.

- **Transmission Rate (Bitrate) $\mathbf{R_b}$**: The instantaneous transmission rate, denoted as R_b, is determined by the MCS table, which depends on the number of spatial streams (NSS) and the bandwidth (B). Mathematically, it is expressed as:

$$R_b = R_{\text{MCS}}(\text{NSS}, B) \cdot \text{NSS}, \tag{3.9}$$

where $R_{\text{MCS}}(\text{NSS}, B)$ represents the transmission rate per spatial stream.

- **Received Signal Strength Indicator (RSSI) $\mathbf{\Gamma}$**: A vector representing the received signal strength for each antenna at time t, expressed as:

$$\mathbf{\Gamma} = [\Gamma_1, \Gamma_2, \ldots, \Gamma_L], \tag{3.10}$$

where $\Gamma_i(t)$ denotes the RSSI measured at the i-th antenna, and N_{ant} is the total number of antennas.

- **Application-Level Throughput η**: The throughput available at the application level is defined as the achieved throughput at time step t, given by:

$$\eta_t = \frac{B_t}{t} \tag{3.11}$$

where B_t represents the number of successfully transmitted bytes correctly transmitted at the Application Layer during time interval t.

To estimate the variability of the channel, we compute the variance of each KPI over a temporal window W. Specifically, for a KPI $K(t)$, the variance is defined as:

$$V_K(t) = \frac{1}{W} \sum_{k=t-W+1}^{t} \left(K(k) - \mu_K(t)\right)^2 \tag{3.12}$$

where $\mu_K(t)$ is the mean of KPI $K(t)$ over the window W. This variance serves as an indirect measure of the channel's stability, with higher variance indicating more rapid changes and shorter coherence times, further details will be presented in Section 3.4.

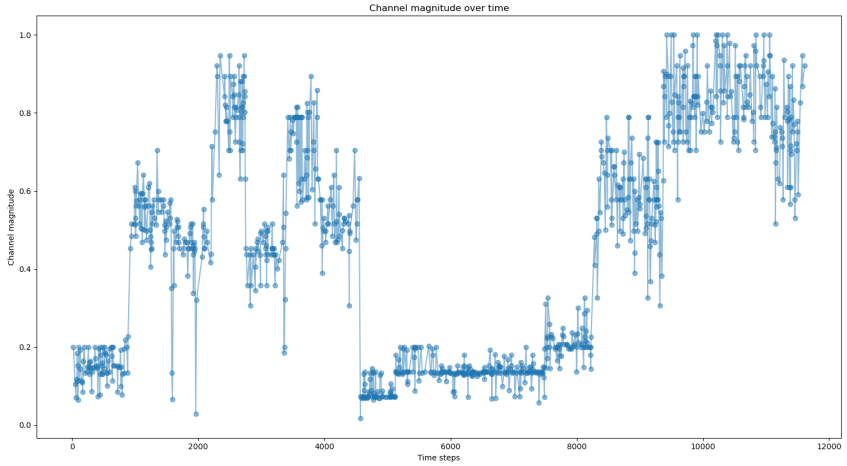


Figure 3.3: Channel magnitude over time, illustrating the variability of channel conditions and the necessity for frequent antenna selection.

3.3 Problem Formulation

We aim to apply DRL to address the antenna selection problem. This approach is particularly well-suited to the task due to the sequential nature of the problem, where decisions at each time step influence future system states

and performance. To enable the application of DRL, the problem must be formulated as a Discrete Time MDP, defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$. The channel is sampled at regular intervals of duration T_s, during which a channel measurement is performed, an action is taken, and a reward is received. The elements of the MDP are defined as follows:

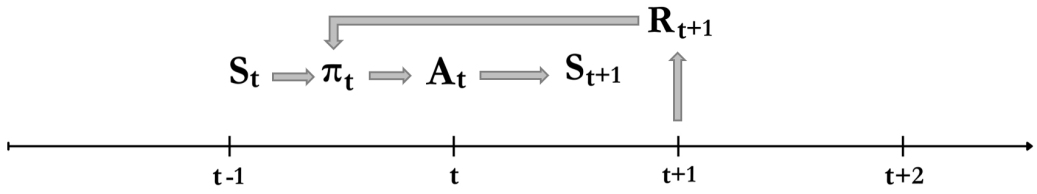


Figure 3.4: Illustration of the temporal view of a MDP.

1. *State Space* $\mathcal{S}$: The state space $\mathcal{S}$ captures the essential features of the system at each time step and is composed of multiple components that represent both the system's current configuration and the observed channel conditions. The state at time t is defined as:

$$s_t = [\mathbf{a}_t, \mathbf{F}_{\text{CFR}}(t), \mathbf{\Gamma}(t), R_b(t), \mathcal{V}(t)], \tag{3.13}$$

where each component is detailed as follows:

- Antenna Configuration $\mathbf{a}_t$

  The antenna configuration at time t is represented as a one-hot encoded vector $\mathbf{a}_t \in \mathbb{R}^M$, where M is the total number of possible

antenna configurations. Formally:

$$\mathbf{a}_t = [a_1, a_2, \ldots, a_M], \quad \text{where} \quad a_i = \begin{cases} 1 & \text{if configuration } i \text{ is active at time } t, \\ 0 & \text{otherwise.} \end{cases} \tag{3.14}$$

- CFR Features $\mathbf{F}_{\text{CFR}}(t)$

  To extract meaningful features from the CFR matrix, we compute its Singular Value Decomposition (SVD). The CFR matrix at time t, denoted as $\mathbf{H}(t) \in \mathbb{C}^{N_{\text{ant}} \times N_{\text{sub}}}$, where N_{ant} is the number of antennas and N_{sub} is the number of subcarriers, is decomposed as:

$$\mathbf{H}(t) = \mathbf{U}(t)\mathbf{\Sigma}(t)\mathbf{V}^H(t), \tag{3.15}$$

  where:

  - $\mathbf{U}(t)$: Left singular vectors,
  - $\mathbf{\Sigma}(t)$: Diagonal matrix of singular values,
  - $\mathbf{V}^H(t)$: Hermitian transpose of the right singular vectors.

  The singular values $\mathbf{\Sigma}(t)$ are used as features, forming the vector:

$$\mathbf{F}_{\text{CFR}}(t) = [\sigma_1(t), \sigma_2(t), \ldots, \sigma_L(t)], \tag{3.16}$$

  where $L = \min(N_{\text{ant}}, N_{\text{sub}})$ is the number of singular values. These values capture the energy distribution of the channel across its principal components and provide insight into the channel's quality and capacity.

- RSSI Vector $\mathbf{\Gamma}(t)$

  The RSSI at time t is represented as a vector $\mathbf{\Gamma}(t) = [\Gamma_1(t), \Gamma_2(t), \ldots, \Gamma_{N_{\text{ant}}}(t)]$, where each element $\Gamma_i(t)$ corresponds to the power (in dB) measured at the i-th antenna. The RSSI provides a measure of the instantaneous signal strength, which is crucial for assessing the quality of the received signal.

- Bitrate $R_b(t)$

  The bitrate $R_b(t)$ represents the instantaneous data transmission rate at time t. It reflects the current channel conditions and depemd on the MCS in use, see Table 1.3. $R_b(t)$ is measured directly from the system, providing a real-time estimate of the communication throughput.

- Channel Variability Conditions $\mathcal{V}(t)$

  To capture the channel's dynamic behavior across multiple timescales, we define $\mathcal{V}(t)$ as the variance of key metrics computed over three temporal windows: short-term (T_s), mid-term (T_m), and long-term (T_l). This approach provides a multi-scale assessment of the channel's stability and variability. The channel variability vector $\mathcal{V}(t)$ is composed of variability metrics across different time ranges:

$$\mathcal{V}_{\text{short}} = [\text{Var}_{T_s}(\mathbf{F}_{\text{CFR}}), \text{Var}_{T_s}(\mathbf{\Gamma}), \text{Var}_{T_s}(R_b)], \tag{3.17}$$

$$\mathcal{V}_{\text{mid}} = [\text{Var}_{T_m}(\mathbf{F}_{\text{CFR}}), \text{Var}_{T_m}(\mathbf{\Gamma}), \text{Var}_{T_m}(R_b)], \tag{3.18}$$

$$\mathcal{V}_{\text{long}} = [\text{Var}_{T_l}(\mathbf{F}_{\text{CFR}}), \text{Var}_{T_l}(\mathbf{\Gamma}), \text{Var}_{T_l}(R_b)]. \tag{3.19}$$

Thus, $\mathcal{V}(t)$ is given by:

$$\mathcal{V}(t) = [\mathcal{V}_{\text{short}}, \mathcal{V}_{\text{mid}}, \mathcal{V}_{\text{long}}]. \tag{3.20}$$

The complete state vector s_t at time t is expressed as:

$$s_t = \Bigg[\underbrace{\mathbf{a}_t,\ \sigma_1(t), \sigma_2(t), \ldots, \sigma_L(t),\ \Gamma_1(t), \Gamma_2(t), \ldots, \Gamma_{N_{\text{ant}}}(t),\ R_b(t)}_{\text{Current Metrics}}, \quad \underbrace{\mathcal{V}(t)}_{\text{System Stability}}\Bigg]^T \tag{3.21}$$

The final state is represented as a column vector, where each component is appropriately normalized to ensure a homogeneous distribution of values within the state space.

2. *Action Space* $\mathcal{A}$: The action $a_t \in \mathcal{A}$ corresponds to selecting an antenna configuration from a finite set:

$$\mathcal{A} = \{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_M\}, \tag{3.22}$$

where M is the total number of possible configurations.

3. *Transition Probability* $\mathcal{P}$: The transition probability $\mathcal{P}(s_{t+1} \mid s_t, a_t)$ represents the likelihood of transitioning to a new state s_{t+1} given the current state s_t and the action a_t. In this real-world antenna selection scenario, the transition dynamics are inherently stochastic and

unknown due to the complexity and variability of the wireless environment. No explicit model is applied to the environment, and the agent learns these dynamics solely through interactions with the system. This formulation allows the agent to adapt to dynamic and unpredictable channel conditions without requiring prior knowledge of the environment's statistical properties.

4. *Reward Function R*: The immediate reward $r_t = R(s_t, a_t)$ is a measure of the system's performance based on the selected antenna configuration and the observed channel conditions. We will use the application throughput:

$$r_t = \eta_t \tag{3.23}$$

This formulation ensures that the reward reflects the efficiency of data transmission in the system.

5. *Discount Factor* $\gamma \in [0, 1]$: The discount factor γ determines the importance of future rewards relative to immediate rewards. A higher value of γ prioritizes long-term performance.

6. *Policy and Objective Function*: A policy $\boldsymbol{\pi} = \{\pi_0(\cdot), \pi_1(\cdot), \dots\}$ is defined as a sequence of decision rules $\pi_t(\cdot)$, where each rule at time t maps the state vector s_t to an action a_t, such that:

$$a_t = \pi_t(s_t). \tag{3.24}$$

Given an initial belief vector s_0, the objective function $J_\pi(s_0)$ for an

finite time horizon T is defined as:

$$J_\pi(s_0) = \mathbb{E}\left[\sum_{t=0}^{T} R(s_t, a_t) \mid s_0\right], \tag{3.25}$$

where $\mathbb{E}_{\{s_t\}}$ represents the expectation with respect to the joint probability distribution of $\{s_t\}_{t=0}^{\infty}$, given the initial distribution b_0. Since $a_t = \pi(b_t)$, the objective function is parameterized by the policy $\pi(\cdot)$, which is reflected in the subscript π. The goal is to determine the optimal policy π^* that maximizes the objective function:

$$\pi^* = \arg\max_{\pi} J_\pi(s_0), \quad \forall s_0. \tag{3.26}$$

3.4 Deep Reinforcement Learning based Antenna selection

3.4.1 Processing Pipeline

The processing pipeline of the proposed PPO-based algorithm for antenna selection follows a structured sequence of operations to optimize wireless communication performance. The pipeline begins with traffic generation between the access point (AP) and a client device, ensuring the exchange of packets under an initial antenna configuration. Following traffic generation, the CFR and other KPIs are acquired using the Broadcom utility tool. These raw measurements serve as the foundation for the RL framework, forming the state representation and reward signal.

Once collected, the acquired data undergo a preprocessing phase to ensure consistency and reliability. Specifically, the CFR features, as discussed in Section 1, are extracted to capture the most relevant channel characteristics. Additionally, anomalous or missing data points are removed to prevent distortions in the learning process. The remaining features are then normalized and concatenated to form a well-structured state vector.

Rather than using individual states in isolation, the algorithm constructs a temporal sequence of states before feeding them into the model. This approach enhances state representation by integrating temporal dependencies, which are crucial for capturing the channels dynamic variations and improving decision-making.

The reinforcement learning model is based on the PPO framework, as detailed in Section 2.6. It consists of two neural networks: the policy network and the value network, both of which share the same architecture. These networks take as input the previously constructed state sequence. The input is first processed through a GRU layer, which captures sequential dependencies and extracts temporal features. The output of the GRU is then passed through fully connected dense layers, refining the representation for decision-making.

The policy network outputs a probability distribution over actions, determining the next antenna configuration to be selected. Meanwhile, the value network estimates the current state value, providing a baseline for computing advantage estimates during training.

Once the model produces its outputs, the policy is used to select the next action, determining the optimal antenna configuration to apply. Concur-

rently, the state value is stored for subsequent updates. The newly selected antenna pattern is set, and a new state is generated, initiating the next iteration of the process.

At predefined intervals, the update process is paused to perform policy optimization, following the PPO algorithm outlined in (3). During this step, stored experiences are used to update the model parameters, refining the policy and improving future decisions.

This pipeline ensures an efficient and adaptive antenna selection mechanism, leveraging reinforcement learning to dynamically adjust to changing wireless environments while maximizing communication performance.

3.4.2 Signals and Features

Once all the CFR and KPIs metrics have been collected, as detailed in Section 3.3, they undergo a series of processing steps to construct the state representation for the reinforcement learning framework.

First, we extract features from the CFR using SVD 1, obtaining a reduced representation that captures the dominant characteristics of the channel response. This process ensures dimensionality reduction while preserving the most informative components for antenna selection.

Next, missing data points are handled to mitigate measurement inconsistencies. This includes interpolating small gaps and discarding severely corrupted data points that could negatively impact learning. Additionally, the stability metric, introduced in Section 1, is computed to quantify the variability of channel conditions over time.

To ensure numerical consistency across different feature types, all extracted features, including CFR-based singular values, RSSI, bitrate, and stability metrics, are normalized. This step makes the data homogeneous, preventing certain features from dominating the learning process due to scale differences.

After preprocessing, the final state representation is defined as a feature vector of size $\mathbf{S} \in \mathbb{R}^{1\times S}$, where S represents the total number of extracted and preprocessed features.

To enhance the temporal representation of the state, the model processes variable-length sequences of past states. At the beginning of each episode, the sequence starts with a single state. As new states are collected, the historical sequence progressively grows until reaching a fixed upper limit N. Once this limit is reached, the oldest state in the sequence is discarded to maintain a rolling window of the most recent N states. This ensures that the model retains relevant past information while preventing excessive memory growth. The extended state representation is defined as:

$$\mathbf{S'}_t = [\mathbf{S}_{t-N+1}, \mathbf{S}_{t-N+2}, \ldots, \mathbf{S}_t], \quad \mathbf{S'}_t \in \mathbb{R}^{N\times S}. \tag{3.27}$$

Despite processing variable-length sequences, the model always refers to the most recent state $\mathbf{S}_t$ for decision-making. The concatenated sequence serves solely to enrich the representation of $\mathbf{S}_t$, providing additional temporal context. Consequently, both the policy network and the value network compute their respective outputs based on $\mathbf{S}_t$, leveraging the augmented temporal information to improve action selection and state evaluation.

3.4.3 Model Architecture

The proposed model consists of two neural networks: the policy network and the value network. While these networks have distinct objectives and parameter sets, they share the same underlying architecture. This section provides a detailed description of the common architecture, outlining how the input is processed through different layers using a formal mathematical formulation. Subsequently, the specific roles and differences between the policy and value networks are discussed.

Recurrent Module

To capture temporal dependencies in the state representation, the first stage of the network employs a recurrent layer based on a gated mechanism. The input to this module is the stacked state sequence $\mathbf{S}' \in \mathbb{R}^{N \times S}$, where:

- N represents the length of the temporal window.
- S is the dimensionality of a single state vector.

The recurrent layer processes this sequence iteratively, maintaining a hidden state $\mathbf{h}_{t'} \in \mathbb{R}^D$ at each time step t', where D is the number of hidden units in the recurrent layer. The computations at each step t' within the input sequence $\mathbf{S}'$ are defined as follows:

1) Update Gate: Determines the proportion of the previous hidden state to retain.

$$\mathbf{z}_{t'} = \sigma\left(\mathbf{W}_z \mathbf{S}_{t'} + \mathbf{U}_z \mathbf{h}_{t'-1} + \mathbf{b}_z\right), \tag{3.28}$$

where $\mathbf{W}_z \in \mathbb{R}^{D \times S}$, $\mathbf{U}_z \in \mathbb{R}^{D \times D}$, and $\mathbf{b}_z \in \mathbb{R}^D$ are trainable parameters.

2) Reset Gate: Controls how much of the past information is forgotten.

$$\mathbf{r}_{t'} = \sigma\left(\mathbf{W}_r \mathbf{S}_{t'} + \mathbf{U}_r \mathbf{h}_{t'-1} + \mathbf{b}_r\right), \tag{3.29}$$

where $\mathbf{W}_r \in \mathbb{R}^{D \times S}$, $\mathbf{U}_r \in \mathbb{R}^{D \times D}$, and $\mathbf{b}_r \in \mathbb{R}^D$.

3) Candidate Hidden State: Computes a new candidate state based on the reset gate.

$$\tilde{\mathbf{h}}_{t'} = \tanh\left(\mathbf{W}_h \mathbf{S}_{t'} + \mathbf{U}_h(\mathbf{r}_{t'} \odot \mathbf{h}_{t'-1}) + \mathbf{b}_h\right), \tag{3.30}$$

where $\mathbf{W}_h \in \mathbb{R}^{D \times S}$, $\mathbf{U}_h \in \mathbb{R}^{D \times D}$, and $\mathbf{b}_h \in \mathbb{R}^D$.

4) Final Hidden State Update: Combines the previous hidden state and candidate state using the update gate.

$$\mathbf{h}_{t'} = (1 - \mathbf{z}_{t'}) \odot \mathbf{h}_{t'-1} + \mathbf{z}_{t'} \odot \tilde{\mathbf{h}}_{t'}. \tag{3.31}$$

Once the entire input sequence has been processed, we obtain the sequence of hidden states:

$$\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_N] \in \mathbb{R}^{N \times D}. \tag{3.32}$$

From this sequence, we retain only the final hidden state $\mathbf{h}_N \in \mathbb{R}^D$, which corresponds to the most recent state $\mathbf{S}_t$ in the sequence. In Figure 3.5 an illustration of a GRU is shown.

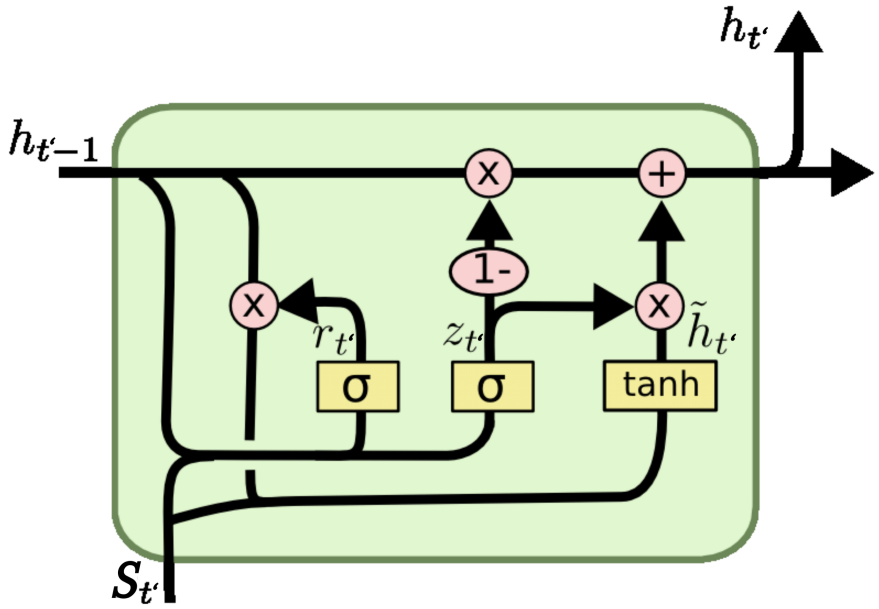


Figure 3.5: Illustration of a Gated Recurrent Unit

Fully Connected Network

The final hidden state $\mathbf{h}_N$ serves as the input to a series of fully connected layers, which refine the representation for policy and value estimation. The first fully connected layer is:

$$\mathbf{F}^{(1)} = \sigma\left(\mathbf{W}^{(1)}\mathbf{h}_N + \mathbf{b}^{(1)}\right), \tag{3.33}$$

where:

- $\mathbf{F}^{(1)} \in \mathbb{R}^{D_1}$ is the output of the first dense layer.
- $\mathbf{W}^{(1)} \in \mathbb{R}^{D_1 \times D}$ is the weight matrix.
- $\mathbf{b}^{(1)} \in \mathbb{R}^{D_1}$ is the bias term.
- $\sigma(\cdot)$ is the Rectified Linear Unit (ReLU) activation function.

Subsequent layers follow the same structure, applying dropout regularization:

$$\mathbf{F}^{(l)} = \text{Dropout}(\sigma(\mathbf{W}^{(l)}\mathbf{F}^{(l-1)} + \mathbf{b}^{(l)}), p), \tag{3.34}$$

where:

- $\mathbf{F}^{(l)} \in \mathbb{R}^{D_l}$ is the output of the l-th dense layer.
- $\mathbf{W}^{(l)} \in \mathbb{R}^{D_l \times D_{l-1}}$ is the weight matrix.
- $\mathbf{b}^{(l)} \in \mathbb{R}^{D_l}$ is the bias term.
- p is the dropout rate.

Policy and Value Networks

The final layer differs for the policy and value networks.

Policy Network: The policy network outputs a probability distribution over the action space, computed through a final dense layer followed by the softmax function:

$$\mathbf{f}_\pi = \mathbf{W}_\pi \mathbf{F}^{(L)} + \mathbf{b}_\pi, \quad \mathbf{f}_\pi \in \mathbb{R}^{|\mathcal{A}|}. \tag{3.35}$$

where $\mathbf{W}_\pi \in \mathbb{R}^{|\mathcal{A}| \times D_L}$ is the weight matrix, D_L is the output dimension of the last dense layer and $b_\pi \in \mathbb{R}^{|\mathcal{A}|}$ is the bias term.

The action probabilities are obtained as:

$$\pi_\theta(a|\mathbf{S}_t) = \frac{\exp(f_{\pi,a})}{\sum_{a' \in \mathcal{A}} \exp(f_{\pi,a'})}, \quad a \in \mathcal{A}, \tag{3.36}$$

where:

- $\mathbf{f}_\pi \in \mathbb{R}^{|\mathcal{A}|}$ represents the logits for each action.
- $f_{\pi,a}$ is the scalar output associated with action a.

Value Network: The value network produces a single scalar estimate of the value function:

$$V_\phi(\mathbf{S}_t) = \mathbf{W}_V \mathbf{F}^{(L)} + b_V, \tag{3.37}$$

where $\mathbf{W}_V \in \mathbb{R}^{1 \times D_L}$ is the weight matrix, D_L is the output dimension of the last dense layer and $b_V \in \mathbb{R}$ is the bias term.

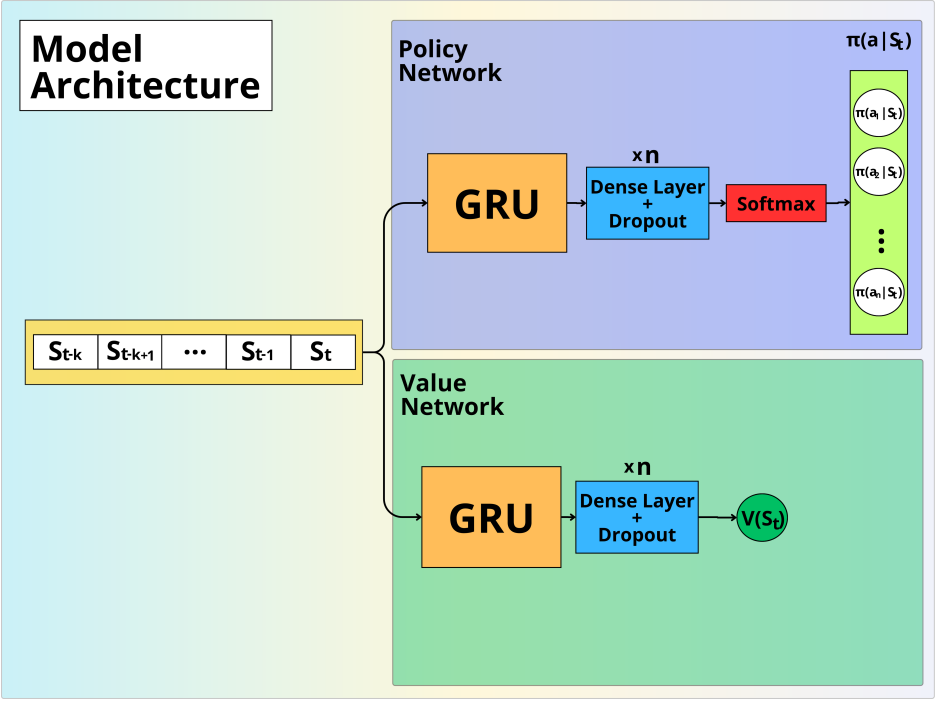


Figure 3.6: Policy and Value Network.

3.4.4 Training Process

The training process in PPO alternates between two main phases: data collection and policy update. The objective is to first collect a batch of experience trajectories and then use them to optimize both the policy and value function.

Data Collection

During the data collection phase, the agent interacts with the environment over a fixed period T_{episode}, gathering experience in the form of state transitions. At each time step t, the following sequence of operations is performed:

1. The agent samples an action a_t from the current policy $\pi_{\theta_{\text{old}}}$:

$$a_t \sim \pi_{\theta_{\text{old}}}(a|S_t). \tag{3.38}$$

2. The selected action is executed in the environment, leading to the next state S_{t+1} and corresponding reward r_t.

3. The estimated value of the current state is computed using the value network:

$$v_t = V_\phi(S_t). \tag{3.39}$$

4. The transition tuple $(S_t, a_t, r_t, S_{t+1}, v_t)$ is stored in memory.

5. This process continues for T_{episode} time steps, accumulating a complete

trajectory:

$$\tau = \{(S_0, a_0, r_0, S_1, v_0), (S_1, a_1, r_1, S_2, v_1), \ldots, (S_T, a_T, r_T, S_{T+1}, v_T)\}. \tag{3.40}$$

Unlike conventional methods that use a fixed-length sequence of past states, our approach dynamically accumulates historical transitions up to time t. Initially, only the single state S_t is used, but as the episode progresses, the model gains access to longer state sequences, leveraging the recurrent structure of the network to encode temporal dependencies.

Policy Update

Once a batch of trajectories is collected, the algorithm proceeds with the policy update. This involves computing the GAE and optimizing both the policy and value networks.

1. The Generalized Advantage Estimation $\hat{A}_t$ is computed following the definition in Algorithm 1.

2. The policy network is updated by maximizing the clipped surrogate objective $\mathcal{L}_{\text{CLIP}}$, as defined in Equation 2.80. To encourage exploration, an entropy bonus term $\mathcal{H}$ (2.81) is also included in the final objective:

$$\mathcal{L}_\pi(\theta) = \mathcal{L}_{\text{CLIP}}(\theta) + c_H \mathcal{H}(\theta). \tag{3.41}$$

3. The value network is updated by minimizing the squared error between the estimated and actual returns, using the loss function $\mathcal{L}_V$

from Equation 2.19.

4. The policy parameters are updated via gradient ascent:

$$\theta \leftarrow \theta + \eta_\pi \nabla_\theta \mathcal{L}_\pi(\theta), \tag{3.42}$$

and the value network is updated via gradient descent:

$$\phi \leftarrow \phi - \eta_V \nabla_\phi \mathcal{L}_V(\phi). \tag{3.43}$$

5. The updated parameters θ and ϕ replace the old ones, and the process is repeated for subsequent training iterations.

Training Iterations

The procedure is iterated for a fixed number of episodes, N_{episode}, independently of the convergence of the policy. The total training time remained constant, as we set a predefined duration instead of waiting for policy convergence. This decision was made because the exact time required for convergence was unknown. Instead, we fixed an arbitrary training duration and evaluated the performance after this period. Consequently, the total training time is given by:

$$T_{\text{training}} = N_{\text{episode}} \times T_{\text{episode}}. \tag{3.44}$$

The details of the proposed training algorithm based on PPO are presented in Algorithm 4 and while in Figure 3.7.

3.4.5 Test Process

Once the training phase is completed, the model is evaluated using a separate test phase to assess its generalization capabilities. Unlike training, where both data collection and policy update steps occur iteratively, the testing phase does not involve policy updates and does not perform the full data collection process. Instead, it only includes the specific part of data collection that consists of inserting the current state or, more precisely, the sequence of the last n states to determine the action to be taken. The pre-trained model is then used to select actions based on this sequence, allowing for an evaluation of its performance on unseen data.

Algorithm 4 Training Algorithm for Antenna Selection Using PPO

Initialize policy network π_θ and value network V_ϕ with parameters θ, ϕ.
Initialize the sampling policy $\pi_{\theta_{\text{old}}}$ with $\theta_{\text{old}} \leftarrow \theta$.
Set optimizer with learning rate α, discount factor γ, GAE parameter λ_{GAE}, and entropy coefficient c_H.
for each episode $e = 1, 2, \ldots, N_{\text{episode}}$ **do**
    Obtain initial state s_0.
    Start timer for the episode with duration T_{episode}.
    while timer not expired **do**
        Construct state sequence $\mathcal{S}_t = [s_0, s_1, \ldots, s_t]$, including all states up to time t.
        Compute action probabilities $\pi_{\theta_{\text{old}}}(a|\mathcal{S}_t)$ and value estimate $V_\phi(\mathcal{S}_t)$ by passing $\mathcal{S}_t$ through the networks.
        Sample action $a_t \sim \pi_{\theta_{\text{old}}}(a|\mathcal{S}_t)$.
        Execute action a_t, observe reward r_t and next state s_{t+1}.
        Store transition $(s_t, a_t, r_t, s_{t+1}, \pi_{\theta_{\text{old}}}(a_t|\mathcal{S}_t)\, V_\phi(\mathcal{S}_t))$ in buffers S, A, R, P, V.
    end while
    Compute advantages $\hat{A}_t$ using GAE.
    Compute policy loss:

$$\mathcal{L}_{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(\rho_t(\theta)\hat{A}_t, \text{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]. \tag{3.45}$$

    Compute value loss:

$$\mathcal{L}_{\text{V}}(\phi) = \mathbb{E}_t \left[\left(V_\phi(\mathcal{S}_t) - R_t \right)^2 \right]. \tag{3.46}$$

    Compute total policy loss:

$$\mathcal{L}_\pi(\theta) = \mathcal{L}_{\text{CLIP}}(\theta) + c_H \mathcal{H}(\theta). \tag{3.47}$$

    Update the policy network parameters:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \mathcal{L}_\pi(\theta). \tag{3.48}$$

    Update the value network parameters:

$$\phi \leftarrow \phi - \alpha \nabla_\phi \mathcal{L}_{\text{V}}(\phi). \tag{3.49}$$

    Synchronize the sampling policy:

$$\theta_{\text{old}} \leftarrow \theta. \tag{3.50}$$

    Clear buffers S, A, R, P, V to prepare for the next episode.
end for
Save the trained policy network π_θ and value network V_ϕ.

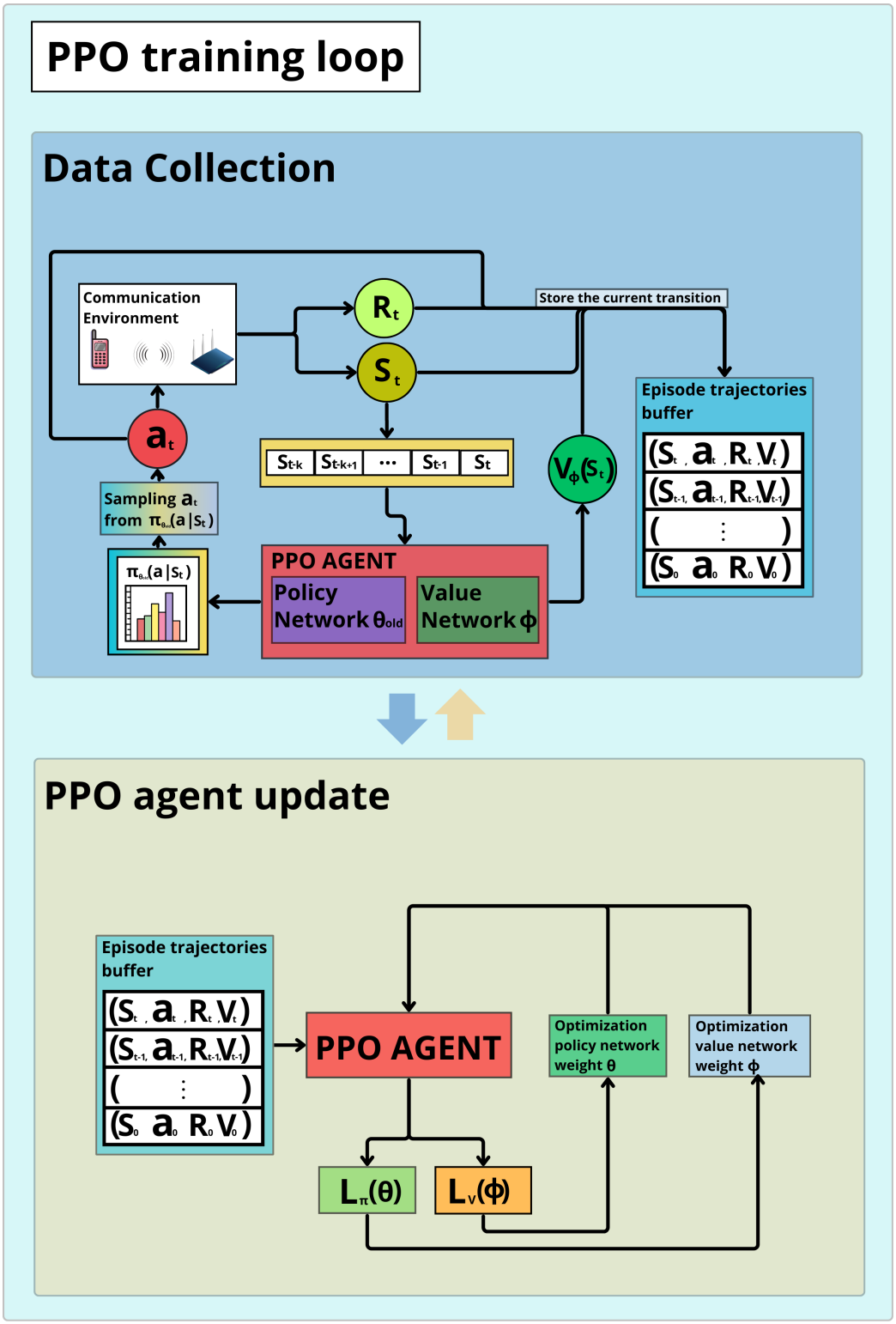


Figure 3.7: Experience Collection and Policy Optimization Cycle

Chapter 4

Performance Evaluation

To assess the performance of our agent, we designed two testing scenarios, comparing its performance with a traditional antenna selection algorithm that does not involve any learning-based approach. The learning environment used for the evaluation consists of an office space with several individuals engaged in different activities and movements. These activities introduce dynamic multi-path fading effects, providing a realistic and challenging testing setup. Within this environment, we deployed our AP connected to a client, which generated data traffic. The specifications of the device used are summarized in Table 4.1.

The two testing scenarios conductedare the following:

1. **Static Tests:** The client remained stationary at a specific location within the office space.

2. **Dynamic Tests:** The client was moved around the office space, simulating a dynamic user scenario.

These tests allow for a comprehensive evaluation of the agent's ability to adapt to both static and dynamic conditions, offering insights into its robustness and effectiveness in real-world scenarios.

4.0.1 Experimental Setup

The testing setup was designed to evaluate the performance of the proposed system under realistic and controlled conditions. The central component of the setup was a WiFi 6E router AP, equipped with a Broadcom radio chip. An application installed on the router allowed dynamic selection of antenna configurations and provided access to KPIs. These KPIs were essential for probing the channel and constructing the channel state representation, as detailed in Section 3.22.

All data collected from the router were transmitted to a MacBook Pro 2020, which acted as the central processing unit. On this machine, the raw data were preprocessed to handle malformed or missing entries that could potentially disrupt the subsequent algorithmic operations. This preprocessing step ensured the robustness and reliability of the input data fed to the reinforcement learning agent.

The MacBook Pro hosted the software implementation of the agent, which was responsible for determining and transmitting the optimal antenna configuration back to the router. Concretely, this step corresponded to executing an action within the reinforcement learning framework.

After the agent's action was executed, the router performed a data-gathering phase to collect updated KPIs. These data were then sent back to

the MacBook Pro, where they were used to form the next channel state and subsequently fed into the PPO algorithm for further decision-making.

Table 4.1: Specifications of the Access Point and Smartphone Used in the Experiment

Parameter	**Specification**
Number of AP RF Chains	4
Operating Frequency	5 GHz
Channel Bandwidth	80 MHz
Antenna for each RF Chain	2
Total Antenna Configurations	16
client	iPhone 11
WiFi Standard	802.11ax
Data Gathering Time	1 s
$T_{episode}$	60 seconds
Training time	60 min

Table 4.2: Optimized PPO Agent Parameters (Using Optuna)

Parameter	**Value**
Discount Factor (γ)	0.9
Clipping Range (ϵ)	0.25
Recurrent Units	128
Entropy Coefficient (c_{entropy})	0.1
Value Loss Coefficient	0.6
Policy Dropout Rate	0.001
Value Dropout Rate	0.001
Learning Rate	8.40×10^{-4}
GAE Lambda (λ)	0.9896
Policy Network Dense Layers	4
Policy Dense Units	224
Value Network Dense Layers	4
Value Dense Units	192
Policy Activation Function	ReLU
Value Activation Function	ReLU

Parameter	**Value**
Optimizer	Adam
Hyperparameter Optimization Method	Optuna

4.0.2 Baseline Policy

The baseline policy is a proprietary antenna selection algorithm designed to optimize the selection of antenna patterns in residential Wi-Fi devices. Its primary objective is to maximize channel quality between the access point and the connected wireless stations, thereby improving connection reliability and overall performance.

The policy operates through two main algorithmic states:

- **Idle phase:** The algorithm maintains the currently selected antenna pattern, which is considered optimal based on previously acquired knowledge.
- **Training phase:** The algorithm explores alternative antenna patterns or re-evaluates existing ones under different channel conditions.

This approach leverages hardware-specific knowledge, including radiation diagrams and antenna placement, to enhance and tailor the selection process. By integrating traditional statistical techniques with system-specific optimizations, the baseline algorithm significantly reduces antenna selection time while ensuring an optimal balance between performance and adaptability.

A key characteristic of the baseline policy is that it does not rely on machine learning techniques, making it particularly well-suited for embedded systems with limited computational resources. Instead, it employs deterministic selection strategies that are optimized for the target hardware.

In the context of this thesis, the baseline policy serves as a benchmark for evaluating alternative antenna selection strategies, including the novel approach proposed in this work. As an established solution, it provides a reference point for assessing key factors such as:

- **Channel quality improvement:** Evaluating the effectiveness of antenna selection in optimizing signal conditions.
- **Environmental responsiveness:** Assessing how well the policy adapts to dynamic wireless environments.
- **Computational efficiency:** Measuring the trade-off between performance and processing overhead.

While the baseline policy excels in computational efficiency and high-performance operation, its reliance on hardware-specific optimizations introduces a trade-off. The algorithm is tightly coupled to the device for which it was designed, limiting its adaptability in highly dynamic environments compared to more generalizable learning-based strategies.

4.0.3 Static Test

The static test scenarios were designed to evaluate the agent's ability to optimize throughput when the smartphone remains stationary at a fixed location. These tests were conducted in the previously described office environment, ensuring that key parameters such as the distance between the AP and the smartphone, as well as the channel conditions, remained unchanged throughout the test duration. This controlled setup enabled a consistent and reliable assessment of the agent's performance in a stable environment.

Each test was carried out under varying conditions and locations, leading to different RSSI values. Therefore, only the relative comparison between the two methods holds significance for performance evaluation. Table 4.3 presents the test results, where the Improvement column indicates the percentage enhancement of the PPO Agent over the Baseline Policy. The percentage improvement is calculated as follows:

$$\text{Improvement } (\%) = \left(\frac{\text{PPO Agent} - \text{Baseline Policy}}{\text{Baseline Policy}} \right) \times 100 \qquad (4.1)$$

Figures 4.1, 4.2, 4.3, and 4.4 illustrate the throughput over time for various test scenarios, while in figure 4.5 we find the average throughput perspective for different test, providing a visual comparison of performance across different conditions.

Table 4.3: Average Throughput Results for Static Test Scenario with Percentage Improvement

Test #	**PPO Agent (Mbps)**	**Baseline Policy (Mbps)**	**Improvement (%)**
1	**565.11**	513.05	10.14%
2	**668.94**	479.12	39.61%
3	**670.80**	626.94	7.00%
4	**315.62**	279.03	13.10%
5	**661.60**	649.95	1.79%
6	**58.65**	49.05	19.58%
7	**168.77**	152.97	10.34%
8	**666.42**	490.29	35.92%
9	**560.80**	549.03	2.15%
10	**670.80**	526.94	27.30%

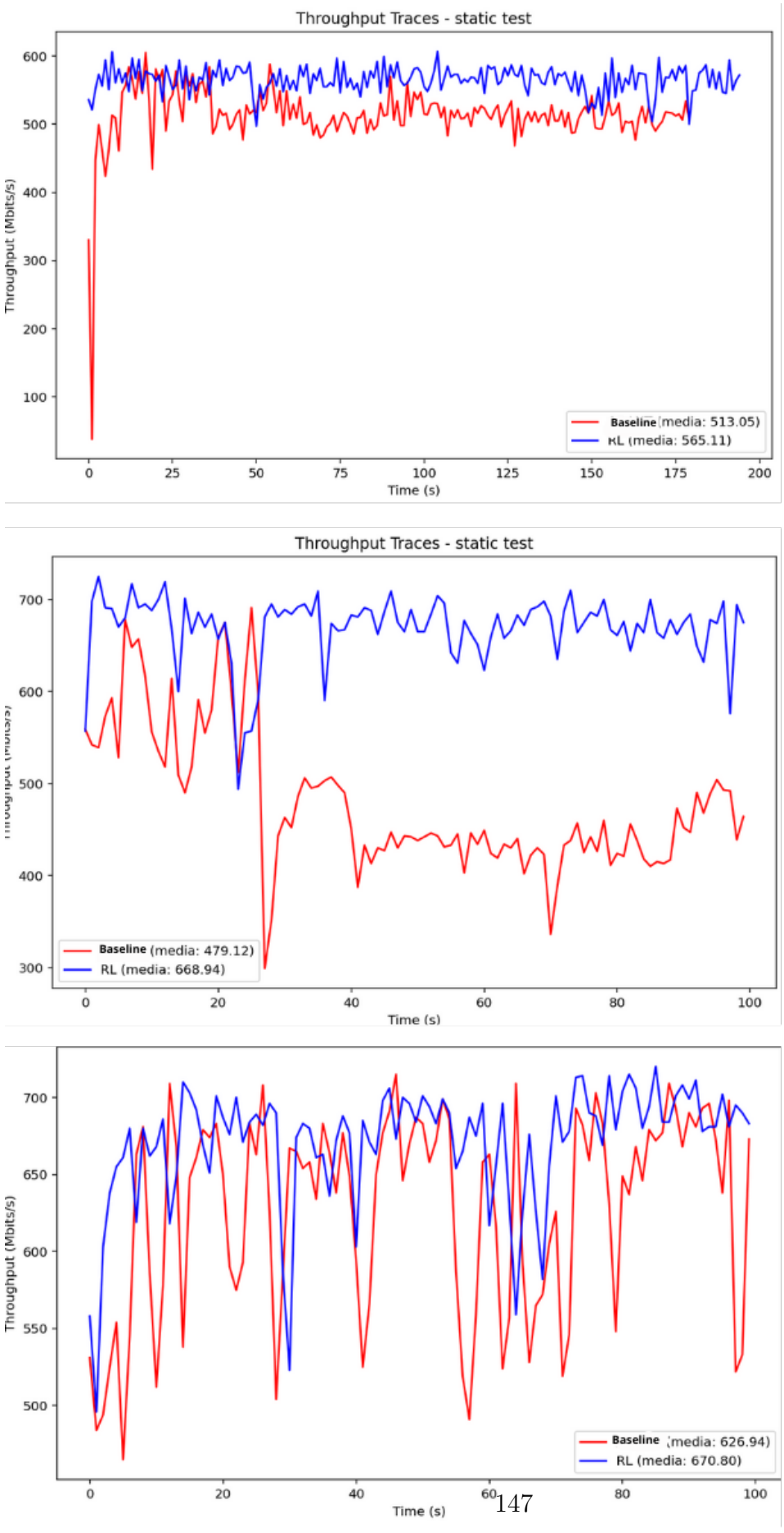


Figure 4.1: Throughput over time for Test 1 (top), Test 2 (middle), and Test 3 (bottom).

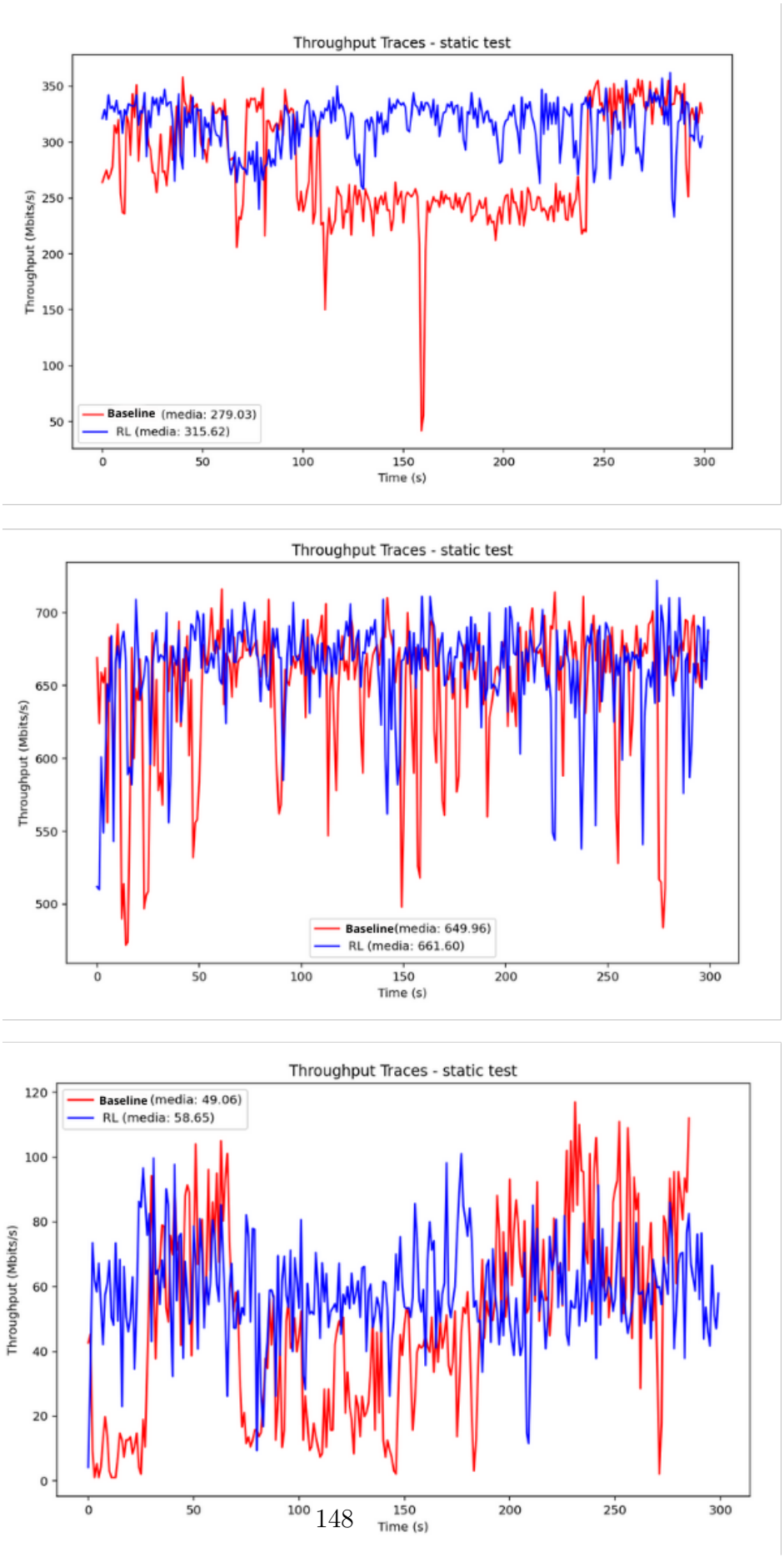
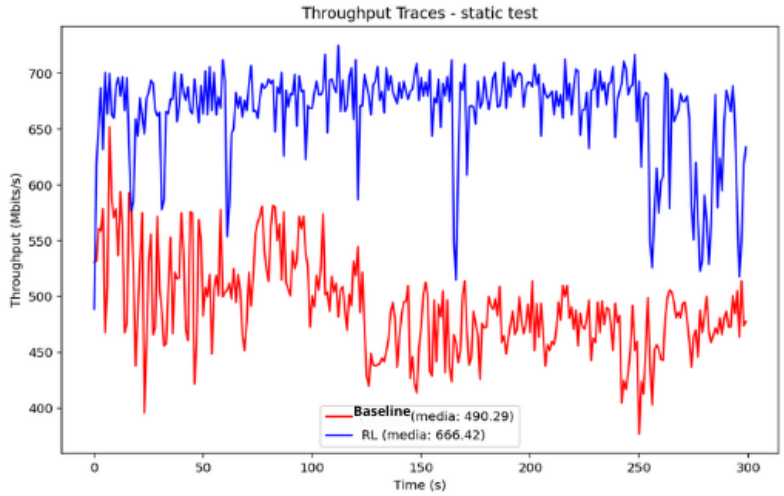


Figure 4.2: Throughput over time for Test 4 (top), Test 5 (middle), and Test 6 (bottom).



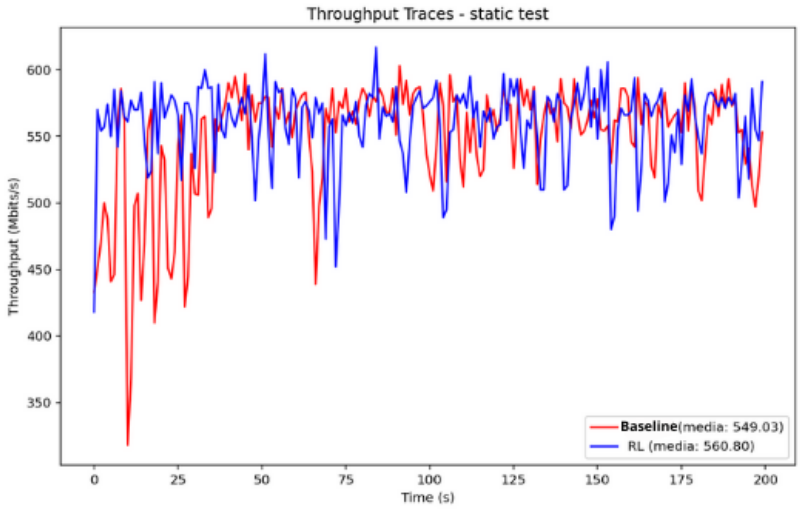


Figure 4.3: Throughput over time for Test 7 (top), Test 8 (middle), and Test 9 (bottom).

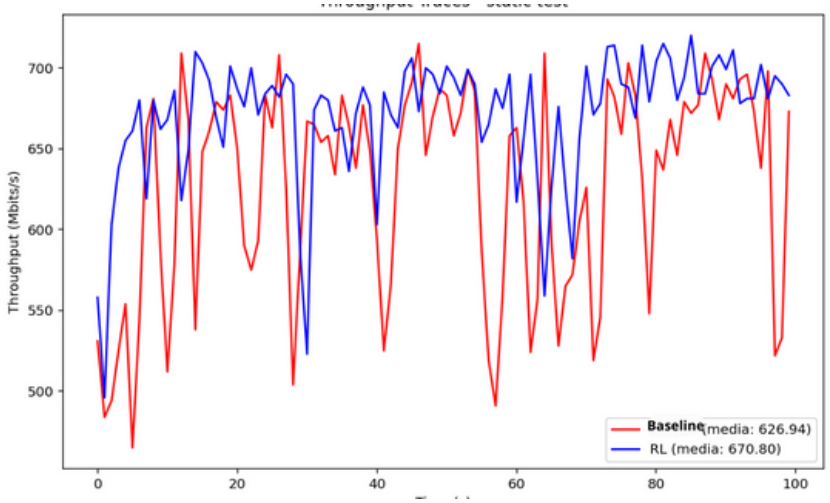


Figure 4.4: Throughput over time for Test 10

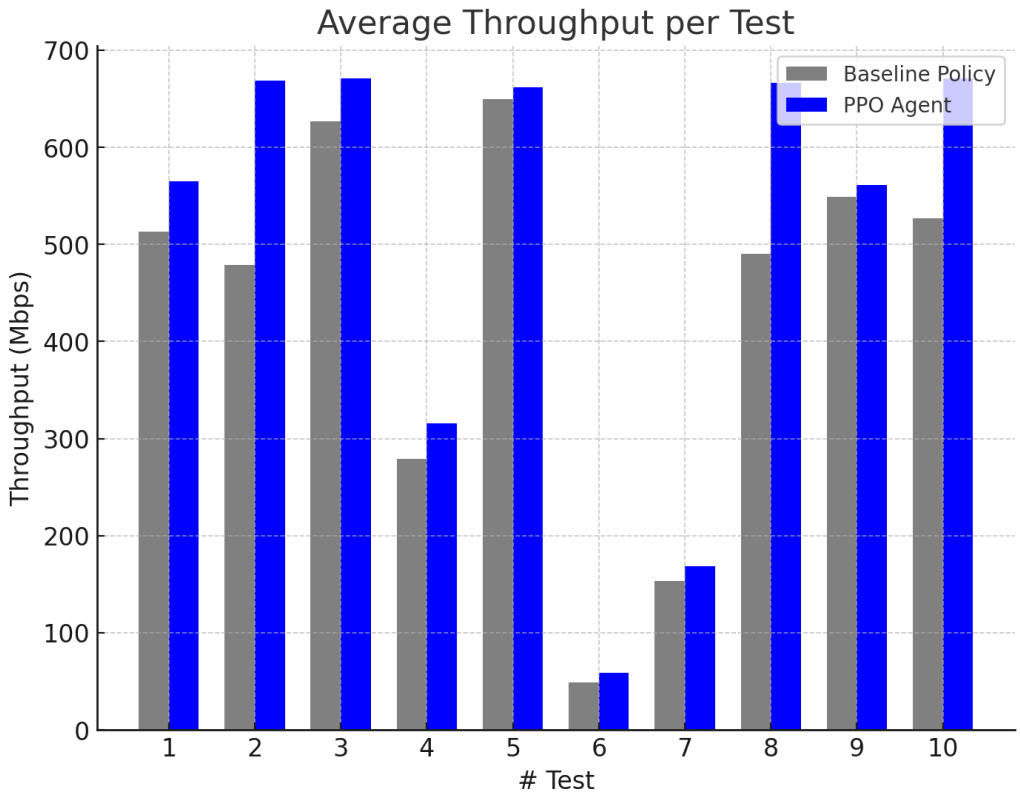


Figure 4.5: Average throughput across different test

4.0.4 Dynamic Tests

For the dynamic tests, we designed a scenarios, referred to as the walking test, to evaluate the performance of the reinforcement learning agent against the baseline policy. The walking test involved traversing the office floor plan in a predefined trajectory. To ensure the test results were comparable over time, we aimed to maintain a consistent walking speed during all test iterations. This consistency allowed for a fair comparison between the reinforcement learning agent and the baseline policy under identical conditions. Both the training and testing phases of the reinforcement learning agent were conducted in the same environment and with the same movement pattern to guarantee experimental validity. Figures 4.6, 4.7 and 4.8 illustrate the throughput over time for various test scenarios, while in figure 4.9 we find the average throughput plot for different test.

Table 4.4: Average Throughput Results for Dynamic Test Scenario with Percentage Improvement

Test #	**RL Agent (Mbps)**	**Baseline Policy (Mbps)**	**Improvement (%)**
1	**295**	245	20.41%
2	**305**	250	22.00%
3	**275**	245	12.24%
4	**305**	230	32.61%
5	**290**	235	23.40%

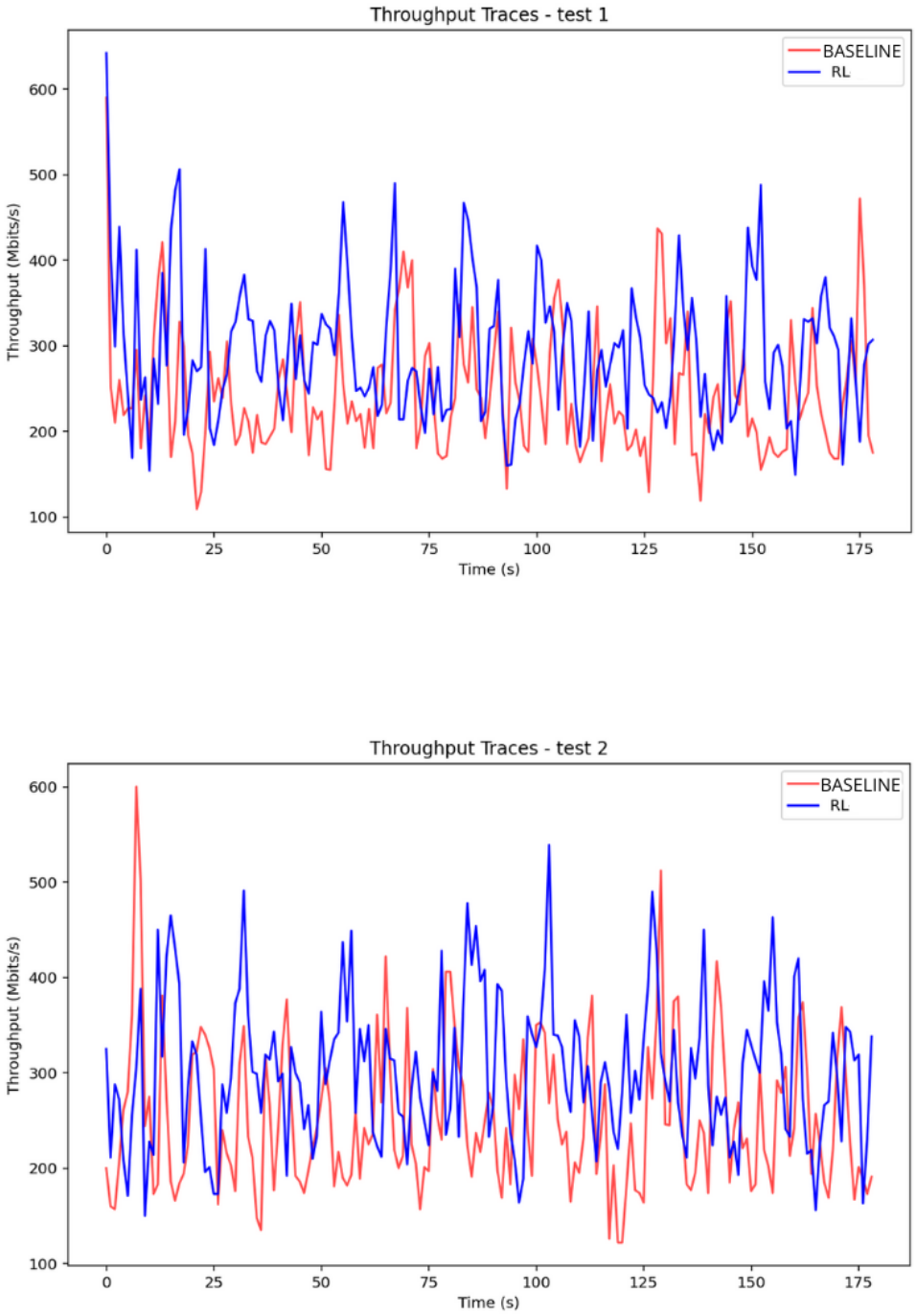


Figure 4.6: Walking Test 1 (top) and Walking Test 2 (bottom).

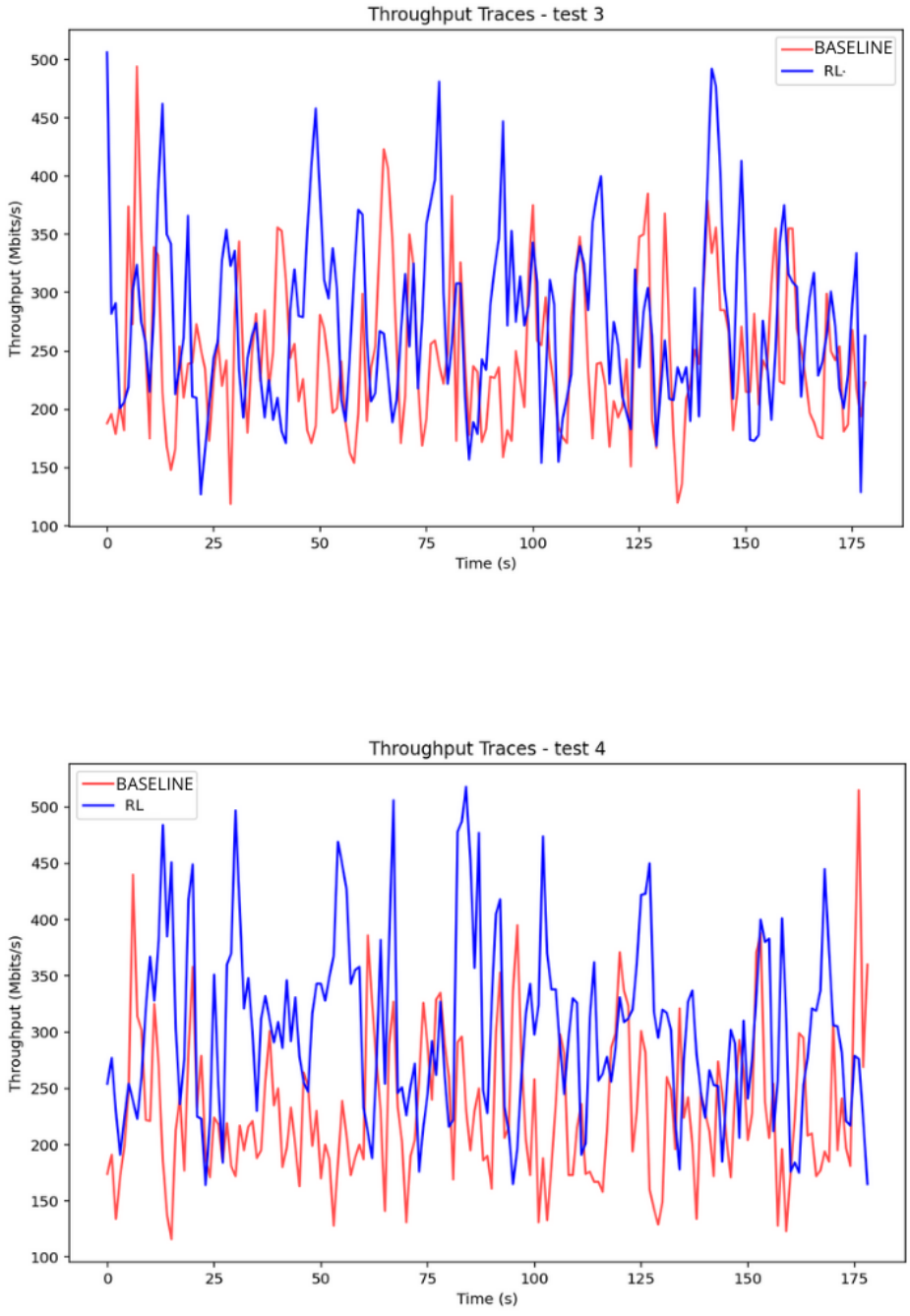


Figure 4.7: Walking Test 3 (top) and Walking Test 4 (bottom).

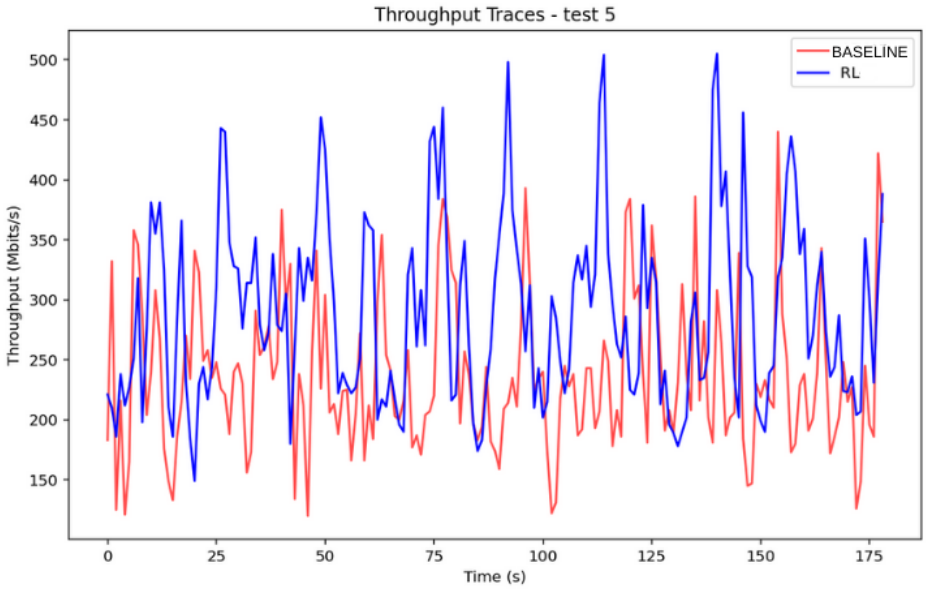


Figure 4.8: Walking Test 5

The graph below shows the average throughput across different tests.

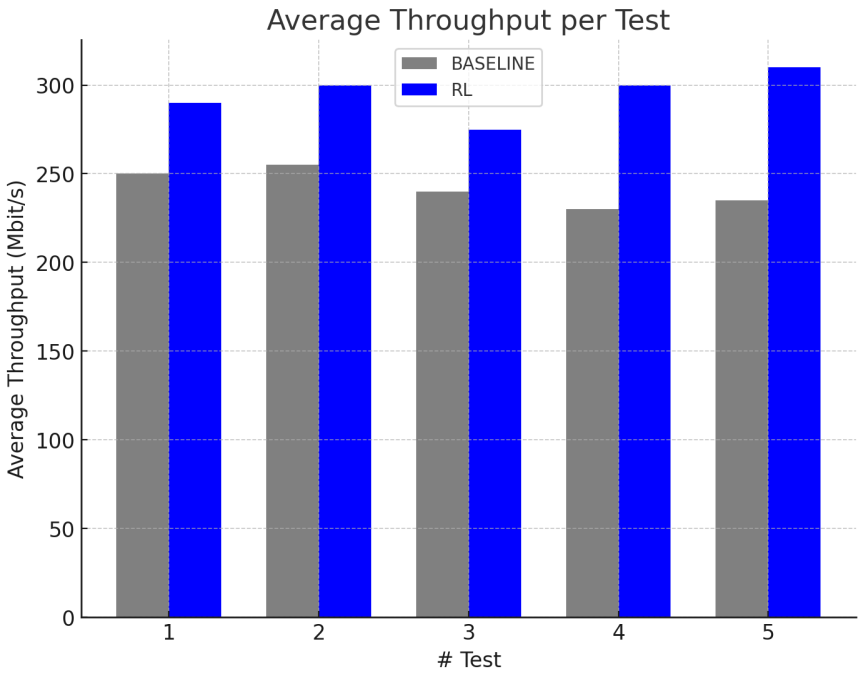


Figure 4.9: Average Throughput across different test.

Chapter 5

Conclusion

This thesis has introduced a novel approach to antenna selection in Wi-Fi networks by leveraging DRL, specifically through PPO. The results obtained from extensive experimentation have demonstrated that reinforcement learning can significantly enhance network performance, surpassing traditional selection strategies and paving the way for intelligent, self-optimizing wireless networks.

5.0.1 Summary of Contributions

The primary contribution of this work is the formulation of the antenna selection problem as a RL task, modeling it as a MDP and defining a state space based on real-time KPIs such as CFR, RSSI, and transmission bitrate. This formulation allowed the learning agent to dynamically adjust the antenna configuration based on continuously evolving channel conditions.

The PPO algorithm was selected due to its balance between stability and efficiency, overcoming the limitations of prior methods such as TRPO. By

integrating a RNN architecture using GRUs, the agent was able to capture temporal dependencies in wireless environments, learning patterns in channel fluctuations that would be impossible to model explicitly. This resulted in a policy that could predict and react proactively to changes in the communication environment, rather than merely responding reactively.

5.0.2 Superior Performance Compared to Traditional Methods

To rigorously evaluate the effectiveness of the proposed RL-based approach, a comprehensive set of experiments was conducted in both static and dynamic environments. The results consistently highlighted the superiority of the reinforcement learning agent:

- In static scenarios, where the wireless client remained at a fixed location, the RL agent consistently selected optimal antenna configurations, achieving higher throughput than the baseline policy. Unlike traditional methods that rely on predefined heuristics or hardware-specific assumptions, the PPO-based approach demonstrated adaptive intelligence, learning directly from the environment without requiring manual tuning.

- In dynamic scenarios, where the wireless client moved across different locations, the advantages of the RL-based method became even more pronounced. The PPO agent exhibited remarkable adaptability, rapidly adjusting antenna configurations in response to real-time variations in the channel. This resulted in a substantial throughput

improvement, confirming that reinforcement learning can effectively generalize across different mobility patterns and environmental conditions.

These findings confirm that machine learning-driven wireless optimization is not just a theoretical concept but a practical, deployable solution that can revolutionize real-world network performance.

5.0.3 Key Strengths and Breakthroughs

The success of this approach stems from several critical innovations:

- *Autonomous learning and adaptation:* Unlike traditional antenna selection algorithms that rely on static rules or handcrafted models, the RL-based approach autonomously learns optimal configurations based on real-world data.

- *Temporal awareness via recurrent neural networks:* The GRU-based architecture enabled the agent to remember past channel states, allowing it to make more informed and predictive decisions rather than relying solely on instantaneous observations.

- *Robustness to environmental variability:* The proposed method demonstrated resilience to fluctuating network conditions, a critical advantage over traditional approaches that require frequent recalibration and manual optimization.

- *Scalability to complex Wi-Fi scenarios:* The algorithm is device-agnostic,

meaning it does not depend on a specific hardware implementation, making it applicable across a broad range of Wi-Fi deployments.

5.0.4 Limitations and Future Research Directions

While the results achieved in this thesis are highly promising, several challenges and opportunities for improvement remain:

1. *Generalization to new environments:* The model was trained and evaluated within a specific testbed environment. While it demonstrated strong adaptability, its ability to generalize across different Wi-Fi deployments, interference scenarios, and network loads remains an open question. Future work should explore domain adaptation techniques and meta-learning approaches to further enhance the models robustness.

2. *Real-time computational efficiency:* Although PPO is significantly more computationally efficient than TRPO, further optimizations could make it more suitable for real-time embedded systems. Techniques such as model compression, quantization, and pruning could be explored to reduce inference latency and enable deployment on resource-constrained devices.

3. *Multi-agent reinforcement learning:* This thesis focused on a single-agent RL approach, where each access point optimizes its own antenna selection. However, in dense wireless environments, multi-agent learning strategies could be implemented, where neighboring APs coordinate

their selection policies to reduce interference and maximize network-wide throughput.

4. *Alternative RL architectures:* While PPO has demonstrated strong performance, future research could explore alternative deep RL algorithms, such as:

    - Soft actor-critic (SAC): This method integrates entropy-based exploration more effectively.
    - Model-based reinforcement learning: Instead of relying purely on experience replay, a model-based approach could learn an internal representation of the environments dynamics, potentially leading to faster convergence and better sample efficiency.

5. *Integration with next-generation wireless standards:* The algorithm developed in this thesis was designed for Wi-Fi 6 (802.11ax). Future work could explore its applicability to Wi-Fi 7 (802.11be) and beyond, where features such as multi-link operation (MLO) and enhanced beamforming could benefit from AI-driven optimization.

5.0.5 Final Remarks

This thesis marks a significant step forward in the application of AI to wireless network optimization. By successfully integrating DRL into the antenna selection process, it has been demonstrated that self-learning algorithms can achieve real-world performance gains, surpassing traditional, manually designed strategies.

The findings presented in this work suggest that machine learning-driven solutions will play an increasingly dominant role in the evolution of future wireless systems. As Wi-Fi networks become more complex, with ever-growing demands for higher data rates, lower latency, and improved spectrum efficiency, AI-powered decision-making will be essential to automate and optimize network operations in real time.

The PPO algorithm, despite being relatively simple, proved to be an effective and robust method for antenna selection, setting the stage for further exploration into more sophisticated AI-driven solutions. The insights gained from this research pave the way for future advancements, ensuring that wireless networks continue to evolve towards greater intelligence, efficiency, and adaptability.

Ultimately, this work lays the foundation for a new era of intelligent wireless communication, where networks are no longer static infrastructures but self-optimizing, data-driven systems capable of real-time adaptation to user demands and environmental conditions.

The appendices presented below provide detailed insights into foundational methods and theoretical concepts that support the research in this thesis.

Appendix A

Reinforcement Learning Basis Methods

This appendix chapter outlines fundamental reinforcement learning (RL) methods. It begins with the Bellman equations, which define recursive relationships in value functions. Then, it introduces key solution methods for MDP, including Dynamic Programming (DP), Monte Carlo methods, and Temporal Difference (TD) learning. These approaches provide the foundation for modern RL algorithms, covering both on-policy and off-policy learning strategies.

A.1 Bellman Equations

A fundamental observation that is used in RL methods is that the value function (Eq. 2.12) can be defined recursively, e.g., we can write the state-value function as follows:

$$Q_\pi(s,a) = \sum_{s' \in \mathcal{S}} P(s'|s,a) \left[r(s,a,s') + \gamma V_\pi(s') \right]. \tag{A.1}$$

Given that $R_t = r_{t+1} + \gamma R_{t+1}$, and putting it together with the equation relating the state-value and state-action value (Eq. 2.14), we can obtain the Bellman equations as follows:

$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(s,a) \sum_{s' \in \mathcal{S}} P(s'|s,a) \left[r(s,a,s') + \gamma V_\pi(s') \right], \tag{A.2}$$

$$Q_\pi(s,a) = \sum_{s' \in \mathcal{S}} P(s'|s,a) \left[r(s,a,s') + \gamma \sum_{a' \in \mathcal{A}} \pi(s',a') Q_\pi(s',a') \right]. \tag{A.3}$$

The Bellman equations express the relationship between the state value/state-action value with its successor states and are used in DP methods to learn the value functions.

A.2 Basic solutions of MDP Problems

A.2.1 Dynamic Programming

Dynamic Programming (DP) is a class of methods to find exact solutions for a conventional MDP problem when the knowledge of the environment dynamic $P(\cdot|s,a)$ is known [2]. Basically, DP learns the value functions (policy evaluation) and uses them to search for a good policy (policy improvement).

Policy Evaluation

It refers to computing the state-value function V_π for an arbitrary policy π. An iterative solution is adopted that uses a successive approximation obtained from the Bellman equation (Eq. A.2). Initially, we start from an arbitrary value of the state-value function V_0 (except terminal state, for which the given value should be 0), and we compute the next state-value function using the following update rule:

$$V_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} P(s'|s,a) \left[r(s,a,s') + \gamma V_k(s') \right], \quad \forall s \in \mathcal{S}. \quad \text{(A.4)}$$

When $k \to \infty$, the convergence of the sequence $\{V_k\}$ is guaranteed under the same conditions that guarantee the existence of the optimal state-value function $V^*(s)$: either $\gamma < 1$ or eventual termination exists starting from any state.

Policy Improvement

It uses the policy evaluation process to find better policies starting from an arbitrary policy π_0. The policy improvement theorem [2] was introduced to compare policies and determine which policy is better than the others. Let π and π' denote any pair of deterministic policies such that for all states $s \in \mathcal{S}$, $V_{\pi'}(s) \geq V_\pi(s)$. Then, the policy π' must be as good as, or better than, π. The new policy is defined by the greedy policy that maximizes the state-action value:

$$\pi'(s) = \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a) \left[r(s,a,s') + \gamma V_\pi(s') \right]. \tag{A.5}$$

Policy Iteration

The combination of policy evaluation with policy improvement in a repeated computation results in policy iteration. Given a policy π_k, we perform policy evaluation to determine V_{π_k}. Then, by using policy improvement, we derive a new policy π_{k+1} which is better than the old policy.

$$V_{k+1}(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a) \left[r(s,a,s') + \gamma V_k(s') \right], \quad \forall s \in \mathcal{S}. \tag{A.6}$$

The update rule is repeated until convergence, i.e., $\max_{s \in \mathcal{S}} |V_{k+1}(s) - V_k(s)| < \epsilon$.

A.2.2 Monte Carlo Sampling

A major limitation of Dynamic Programming (DP) methods is the requirement of knowing the environment dynamics, which may not be available in real-world problems. One approach to address this issue is estimating the state-value function $V(s)$ or the state-action value function $Q(s,a)$ based on samples of transitions from the environment rather than computing them using the transition probability function $P(\cdot|s,a)$. Monte Carlo sampling relies on this idea. The method works as follows:

1. Start from state s_0.

2. Sample a trajectory at time t, $\tau_t = (s_0, a_0, r_1, s_1, a_2, r_3, s_3, \ldots, s_T, a_T, r_{T+1}, s_{T+1})$ from the environment using the current policy π.

3. Compute the discounted cumulative reward:

$$R_t^{(e)} = \sum_{k=0}^{T} \gamma^k r_{t+k+1}, \tag{A.7}$$

for time step t.

4. Repeat the previous steps M times to obtain an estimate:

$$V_\pi(s) \approx \frac{1}{M} \sum_{t=1}^{M} R_t^{(e)}. \tag{A.8}$$

The values of $V_\pi(s)$ are approximated by averaging over multiple trajectories sampled from the environment; however, the exact number of samples required for an accurate approximation is not explicitly determined. In practice, continuous updates of the estimated values are used instead:

$$V_\pi(s) \leftarrow V_\pi(s) + \alpha(R_t - V_\pi(s)), \tag{A.9}$$

where α is a weighting factor (a constant between 0 and 1). The state-action value function $Q(s, a)$ can be estimated similarly:

$$Q_\pi(s, a) \leftarrow Q_\pi(s, a) + \alpha(R_t - Q_\pi(s, a)). \tag{A.10}$$

Monte Carlo sampling is generally used for episodic tasks where trajectories have finite lengths. For continuous tasks, performing reward averaging is

difficult. Furthermore, the method heavily relies on exploitation, so the estimates converge to the optimal values. If actions are sampled greedily from the current policy, the same actions will be chosen, and the agent may get stuck in a suboptimal local minimum. To avoid convergence to suboptimal solutions, randomness is added to the action selection, enforcing exploration of new trajectories to cover a larger range of actions and paths in the estimation process. However, in the case of fully random action selection, the evaluated policy does not correspond to the current policy anymore; rather, it behaves as a random policy. A balance between these two approaches should be maintained for efficient learning. This issue is known as the exploration-exploitation dilemma. A practical approach is to emphasize random selection at the beginning of learning and later shift towards exploiting the learned knowledge.

A.2.3 Temporal Difference methods

In Monte Carlo methods, V_π is estimated using the average return in each episode. However, this technique slows down training and is not feasible for infinite tasks. Formally, RL is a computational approach that enables learning through interaction, where an agent selects an action and receives a numerical reward signal that informs it how good the action is. Instead of learning from expert demonstrations, RL allows an agent to learn autonomously by focusing on maximizing the cumulative reward received from the environment. The outcome of an RL algorithm is a model that maps situations to actions to achieve a defined goal. RL is based on learning behaviors

through interactions with an environment in discrete time steps.

We can draw an analogy with learning to play a new video game: an agent (gamer) tries to choose the best actions in the game (environment) to maximize the score (reward). Therefore, the interaction consists of two major steps: action and perception. At each time step, the agent observes the environment, denoted as o_t, which describes the state s_t from the state space $\mathcal{S}$. Then, according to a policy $\pi(a_t|s_t)$ that dictates the agent's behavior, the agent selects an action a_t from the action space $\mathcal{A}$ and applies it to the environment, leading to a new state s_{t+1}, and so on. The action choice is based on the transition probability $P(s_{t+1}|s_t, a_t)$. Additionally, the environment returns a numerical signal (i.e., reward) $r(s, a)$ as feedback to the agent on the action a_t chosen in the current state s_t. This process is repeated between the agent and the environment until a termination condition is met.

In model-free RL, TD methods are commonly used. These methods replace the average return with an estimate composed of the immediate reward sampled and an estimate of the expected return of the next state:

$$R_t = r(s, a, s') + \gamma R_{t+1} \approx r(s, a, s') + \gamma V_\pi(s'). \tag{A.11}$$

TD methods combine sampling from Monte Carlo with bootstrapping from DP, leveraging the recursive nature of returns. The update rules are given by:

$$V_\pi(s) \leftarrow V_\pi(s) + \alpha\delta, \tag{A.12}$$

$$Q_\pi(s,a) \leftarrow Q_\pi(s,a) + \alpha\delta, \tag{A.13}$$

where δ is defined as:

$$\delta = r(s,a,s') + \gamma Q_\pi(s',a') - Q_\pi(s,a). \tag{A.14}$$

Here, δ, which is called the TD error or reward-prediction error, represents the difference between the immediate reward combined with the expected return of the next state/action and the current reward prediction. The key advantage of TD methods is that they do not require waiting until the end of an episode to update the state-value function or the state-action value function, nor do they necessitate episodic tasks. However, TD methods depend on an estimate that may initially be incorrect and require multiple iterations to start learning effectively.

When learning the Q-function, we distinguish between two different ways of selecting the next action a' in the update rule of Eq. (2.21), i.e., selecting the action using the policy $\pi(s)$, or using the greedy action $a^* = \arg\max_a Q(s',a)$. Two approaches arise, differing in the choice of the next action a':

- **On-policy method**, e.g., SARSA (State-Action-Reward-State-Action): This is a TD method where the next action is sampled using the policy

$\pi(s)$, and the TD error is given by:

$$\delta = r(s, a, s') + \gamma Q_\pi(s', \pi(s')) - Q_\pi(s, a). \tag{A.15}$$

The policy should be ϵ-soft (e.g., ϵ-greedy or softmax) to ensure exploration of different actions.

- **Off-policy method**, e.g., Q-learning: In this case, the greedy action (i.e., the action with the highest Q-value) is selected to update the current value as follows:

$$\delta = r(s, a, s') + \gamma \max_{a'} Q_\pi(s', a') - Q_\pi(s, a). \tag{A.16}$$

The algorithm for Q-learning is outlined in Algorithm 0. The selection of action a can be performed using the ϵ-greedy strategy:

$$\text{action at iteration } k = \begin{cases} \arg\max_{\mathbf{a}\in\mathcal{A}} Q(s, a) & \text{with probability } 1 - \epsilon, \\ \text{random selection} & \text{with probability } \epsilon. \end{cases} \tag{A.17}$$

In several benchmark environments, Q-learning has been shown to converge to the optimal policy even when samples are not optimally collected. It is important to note that the learning rate α needs to decrease over time to prevent the Q-values from fluctuating excessively while still allowing learning

of new, better actions. Additionally, it must satisfy the following conditions:

$$\sum_{t=0}^{\infty} \alpha_t(s,a) = \infty, \tag{A.18}$$

$$\sum_{t=0}^{\infty} \alpha_t^2(s,a) < \infty. \tag{A.19}$$

Algorithm 5 Tabular Q-learning

Initialize $Q_0(s,a) = 0$ for all s, a.
Get initial state s_0.
while not converged, $k = 1, 2, \ldots$ **do**
    Choose action a, get next state s' and reward r.
    if s' is terminal **then**
        $y_{\text{target}} = r$.
        Sample new initial state s'.
    else
        $y_{\text{target}} = r + \gamma \max_{a'} Q_k(s', a')$.
    end if
    Update:

$$Q_{k+1}(s,a) \leftarrow (1-\alpha) Q_k(s,a) + \alpha y_{\text{target}}. \tag{A.20}$$

    Set $s \leftarrow s'$.
end while

Appendix B

Value-Based Reinforcement Learning Models

This appendix chapter explores value-based reinforcement learning models, focusing on deep learning extensions to traditional RL methods. It introduces the DQN, highlighting the challenges of function approximation and how experience replay and target networks stabilize learning. The chapter also covers key advancements such as Double DQN, Prioritized Experience Replay (PER), and Dueling Networks, which improve convergence and efficiency in deep RL models.

B.1 Deep Q-Network

Function approximators are particularly useful for tasks with large or continuous state and action spaces due to their ability to generalize and perform accurately on unseen data. However, employing neural networks as function

approximators in RL introduces several challenges, especially in comparison to supervised learning.

First, RL data (i.e., transitions) can be highly correlated, which can adversely impact the training process by leading the neural network to converge to suboptimal local minima. Moreover, the target values in the loss function for SARSA and Q-Learning can be expressed as:

$$\mathcal{L}(\theta) = \mathbb{E}_{\pi}\left[\left(\underbrace{r(s,a,s') + \gamma Q_{\theta}(s', \pi(s'))}_{\text{target}} - Q_{\theta}(s,a)\right)^2\right], \quad \text{SARSA} \tag{B.1}$$

$$\mathcal{L}(\theta) = \mathbb{E}_{\pi}\left[\left(\underbrace{r(s,a,s') + \gamma \max_{a'} Q_{\theta}(s', a')}_{\text{target}} - Q_{\theta}(s,a)\right)^2\right], \quad \text{Q-Learning} \tag{B.2}$$

are dependent on the function being learned. This stands in contrast to supervised learning setups, such as classification and regression, where target values (i.e., labels) remain constant throughout the training process, allowing the model to be optimized toward fixed targets.

In RL, however, the target value, such as $r(s,a,s') + \gamma \max_{a'} Q_{\pi}(s', a')$ in Q-learning, changes dynamically during training since it relies on the function being approximated. Consequently, this variability introduces instability in the learning process and requires a significantly large number of updates for the neural network to converge. Deep Q-Network (DQN), proposed by Mnih et al. [3], represents an effective implementation of function approximators in

RL, addressing the issues related to correlated inputs/outputs and the non-stationarity of target values. To mitigate the effects of correlated data, DQN introduces a simple yet powerful idea: instead of updating the Q-function sequentially with each transition, the algorithm stores experienced transitions in an extitexperience replay memory or extitreplay buffer with a large capacity (e.g., 10^6 transitions). Subsequently, stochastic gradient descent (SGD) is applied to minibatches sampled randomly from the replay buffer, reducing correlation in training data and stabilizing learning. New transitions replace older ones when the buffer reaches its full capacity. Furthermore, DQN improves the stability of Q-function learning and addresses the non-stationarity of target values $r(s, a, s') + \gamma \max_{a'} Q_\theta(s', a')$ by computing them using a extittarget network, denoted as $Q_{\theta'}$, which is an identical copy of the actual Q-function Q_θ. The target network is updated less frequently (e.g., every several thousand iterations) using the most recent parameters θ. This strategy keeps the target values constant for a sufficient number of iterations, making them more stable. The DQN method is summarized in Algorithm 6.

DQN typically uses a squared loss function; however, alternative loss functions, such as the Huber loss, can be employed to prevent overfitting to outlier targets. The Huber loss function, illustrated in Figure B.1, is defined as:

$$L_\delta(x) = \begin{cases} \frac{1}{2}x^2 & \text{for } |x| < \delta, \\ \delta(|x| - \frac{1}{2}\delta) & \text{otherwise.} \end{cases} \tag{B.5}$$

Additionally, various optimization algorithms, such as Adam, RMSProp,

Algorithm 6 Deep Q-Network (DQN) Algorithm with Experience Replay

1: Initialize replay buffer $\mathcal{B}$ to store transitions.
2: Initialize randomly the Q-function Q_θ and the target network $Q_{\theta'}$ such that $\theta = \theta'$.
3: Observe initial state s_1.
4: **for** $t = 1, \ldots, T$ **do**
5: Select action a_t based on the behavior policy derived from $Q_\theta(s_t, a_t)$ (e.g., ϵ-greedy).
6: Observe new state s_{t+1} and reward r_t.
7: Store transition (s_t, a_t, r_t, s_{t+1}) in replay buffer $\mathcal{B}$.
8: Sample a minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from $\mathcal{B}$.
9: Set target value:

$$y_j = \begin{cases} r_j & \text{if } s_{j+1} \text{ is terminal,} \\ r_j + \gamma \max_{a'} Q_{\theta'}(s_{j+1}, a') & \text{otherwise.} \end{cases} \tag{B.3}$$

10: Optimize loss function:

$$\mathcal{L}(\theta) = \mathcal{L}(Q_\theta(s_j, a_j) - y_j) \tag{B.4}$$

using SGD (or its variants).
11: Every C steps, reset $\theta' = \theta$.
12: **end for**

or vanilla SGD, can be utilized to minimize the loss function $\mathcal{L}(\theta)$.

It is important to emphasize that transitions stored in the replay buffer are sampled uniformly at random for training. As a result, recently collected transitions may not be used immediately and could be incorporated into training at a later stage. Meanwhile, older transitions, including those generated using suboptimal policies, can still be sampled multiple times, contributing to learning stability.

With DQN, the target network is updated less frequently than the learned network, meaning that the target values may be incorrect during the initial phases of training. Incorrect target values can cause the actual Q-function

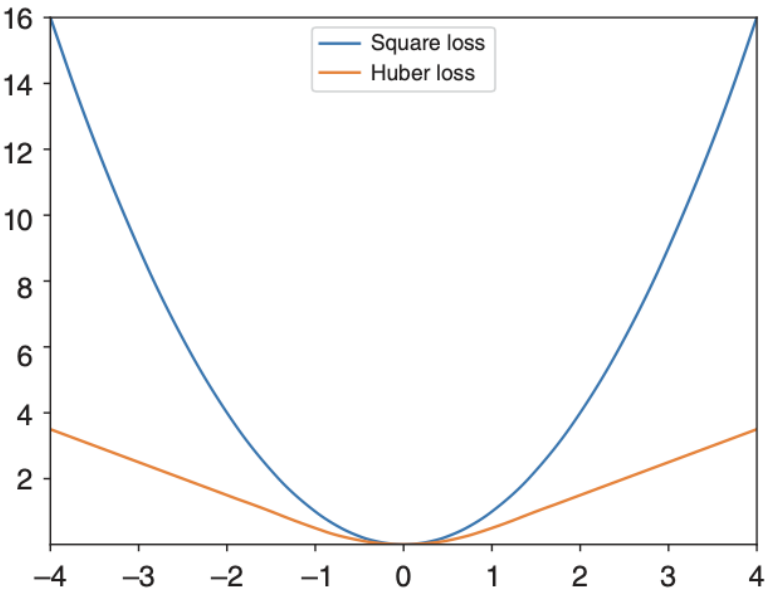


Figure B.1: Comparison between Square Loss and Huber Loss. While square loss penalizes all errors quadratically, Huber loss behaves quadratically for small errors but linearly for large errors, making it more robust to outliers. Reproduced from [5].

to converge toward a suboptimal solution, resulting in poor policy performance. More recent RL algorithms, such as Deep Deterministic Policy Gradient (DDPG), employ Polyak averaging to update the target network as follows:

$$\theta' = \tau\theta + (1 - \tau)\theta', \tag{B.6}$$

where $\tau \ll 1$ is the Polyak parameter. This update method ensures a smooth transition in the target network, allowing it to steadily track the learned Q-function while maintaining stability.

DQN is known to have slow learning and requires a large number of samples to achieve a satisfactory policy. In deep reinforcement learning,

this is commonly referred to as the extitsampling complexity, which denotes the number of environment interactions needed for an algorithm to reach acceptable performance.

B.1.1 Double DQN

Double DQN was proposed by Van Hasselt et al. [20] as an improvement over the standard DQN algorithm. One of the key issues in DQN is the overestimation bias in the target values, which arises due to the use of the max operator in the target update:

$$y = r(s, a, s') + \gamma \max_{a'} Q_{\theta'}(s', a'). \tag{B.7}$$

This overestimation occurs because the Q-values can be inflated, especially in the early stages of training when they are still inaccurate. The max operator selects the action with the highest Q-value from the target network, which may not necessarily be the optimal action but rather one that is overestimated. As a consequence, this leads to an overestimated target value and can result in suboptimal learning.

To mitigate this issue, Double DQN introduces a slight modification: instead of using the target network $Q_{\theta'}$ to select the action in the next state, the action selection is performed using the current Q-network Q_θ, while the evaluation of the Q-value is still done using the target network. The target value is then computed as:

$$y = r(s, a, s') + \gamma Q_{\theta'}\left(s', \arg\max_{a'} Q_\theta(s', a')\right). \tag{B.8}$$

This modification significantly reduces the overestimation bias and improves the stability and performance of the learning process.

B.1.2 Prioritized Experience Replay

A technique that is commonly used in conjunction with the replay buffer is Prioritized Experience Replay (PER) [21]. Standard experience replay samples transitions uniformly, treating all experiences as equally important. However, certain transitions contain more valuable information for learning, and uniform sampling may slow down convergence.

The importance of a transition can be quantified using the temporal difference (TD) error:

$$\delta = r(s, a, s') + \gamma Q_{\theta'}\left(s', \arg\max_{a'} Q_\theta(s', a')\right) - Q_\theta(s, a). \tag{B.9}$$

In greedy TD-error prioritization, transitions are stored along with their TD-error values, and those with the highest TD-error are sampled more frequently. The Q-learning update rule is applied to these transitions, where the gradient is weighted by the TD-error. While this approach improves sample efficiency, it has drawbacks: transitions with initially low TD-error might never be sampled, and excessive prioritization can reduce sample diversity, leading to overfitting.

To address these issues, stochastic sampling is used instead, assigning each transition a probability:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}, \tag{B.10}$$

where $p_i > 0$ is the priority of transition i and α controls the level of prioritization ($\alpha = 0$ corresponds to uniform sampling). The priority p_i can be defined in two ways:

- $p_i = |\delta_i| + \epsilon$, where ϵ is a small positive constant to ensure all transitions have a nonzero probability of being sampled.
- $p_i = \frac{1}{\text{rank}(i)}$, where the rank is determined by sorting transitions according to their TD-error values.

The second approach is more robust to outliers and provides better stability. However, prioritization introduces a bias in the expected value estimation since Q-learning assumes uniform sampling. This bias is corrected using importance sampling (IS) weights:

$$w_i = \left(\frac{1}{NP(i)}\right)^{\beta}, \tag{B.11}$$

where β controls the magnitude of correction, with higher values reducing bias but increasing variance.

To efficiently implement PER, specialized data structures such as sum-trees are used to maintain efficient sampling and updating operations with a complexity of $O(\log N)$.

The training steps using PER are summarized in Algorithm 7.

B.1.3 Dueling Network

The traditional DQN architecture employs a single neural network to directly estimate the Q-values for all possible actions $Q_\theta(s, a)$. However, Baird [22]

proposed decomposing the Q-values into the sum of a state-value function and an advantage function:

$$Q_\pi(s,a) = V_\pi(s) + A_\pi(s,a). \tag{B.12}$$

Here, $V_\pi(s)$ represents the value of the state, while $A_\pi(s,a)$ captures the advantage of selecting action a in that state. A zero advantage corresponds to choosing an optimal action that maximizes the expected return. The advantage of suboptimal actions is negative, making them less favorable. Since advantage values are more constrained than Q-values, neural networks can learn them more effectively.

In [23], the authors proposed integrating this advantage function into the double DQN framework with prioritized replay, as shown in Figure B.2. The key distinction is that the layer before the output separately predicts the advantages $A_{\theta,\alpha}(s,a)$ and state-value $V_{\theta,\beta}(s)$. The Q-value is then computed as:

$$Q_{\theta,\alpha,\beta}(s,a) = V_{\theta,\beta}(s) + A_{\theta,\alpha}(s,a). \tag{B.13}$$

However, V and Q are not uniquely identifiable; adding a constant to A and subtracting it from V leads to the same Q-value. To address this, the advantages are normalized:

$$Q_{\theta,\alpha,\beta}(s,a) = V_{\theta,\beta}(s) + \left(A_{\theta,\alpha}(s,a) - \max_a A_{\theta,\alpha}(s,a)\right). \tag{B.14}$$

A more effective approach replaces the max operator with the mean of

the advantages, improving stability:

$$Q_{\theta,\alpha,\beta} = V_{\theta,\beta}(s) + \left(A_{\theta,\alpha}(s,a) - \frac{1}{|\mathcal{A}|} \sum_{a} A_{\theta,\alpha}(s,a) \right). \tag{B.15}$$

This modification ensures the greedy action has a zero advantage and smooths changes in Q-values as the policy improves.

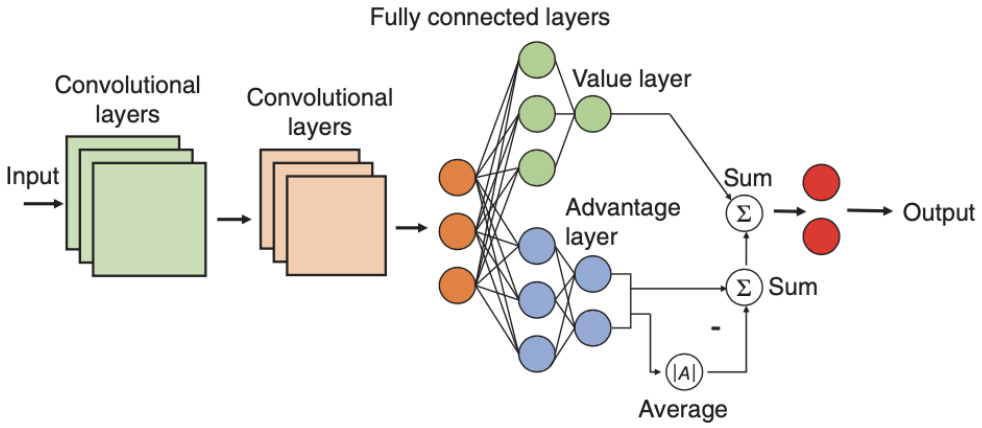


Figure B.2: Dueling network architecture, which separately predicts advantages and state values to compute Q-values. Adapted from [23].

Dueling networks enhance the performance of DQN models, particularly in environments with redundant actions where learning the state value is more beneficial than distinguishing between individual actions.

Algorithm 7 DDQN with prioritized experience replay

Input: Size of minibatch k, learning rate η, update period K, replay size N, exponents α and β, and total number of iterations T.
Initialize replay memory $\mathcal{B} = \emptyset$, weight update accumulator $\Delta = 0$, initial priority $p_0 = 1$.
Observe initial state s_1.
for $t = 1$ to T **do**
    Execute action $a_t \sim \pi_\theta(s_t)$.
    Observe new state s_{t+1} and receive reward r_t.
    Store transition (s_t, a_t, r_t, s_{t+1}) in replay buffer $\mathcal{B}$ with maximal priority $p_t = \max_{i<t} p_i$.
    if $t \bmod K == 0$ **then**
        for $j = 1$ to k **do**
            Sample transition j according to priority distribution:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

            Compute importance sampling weight:

$$w_j = \frac{(NP(j))^{-\beta}}{\max_i w_i}$$

            Compute TD-error:

$$\delta_j = r_j + \gamma Q_{\theta'}(s_{j+1}, \arg\max_{a'} Q_\theta(s_{j+1}, a')) - Q_\theta(s_j, a_j)$$

            Update transition priority: $p_j \leftarrow |\delta_j|$.
            Accumulate weight-change:

$$\Delta \leftarrow \Delta + w_j \delta_j \nabla_\theta Q_\theta(s_j, a_j)$$

        end for
        Update weights: $\theta \leftarrow \theta + \eta \cdot \Delta$, then reset $\Delta = 0$.
        Periodically update target network weights: $\theta' \leftarrow \theta$.
    end if
end for

Appendix C

Actor-Critic Methods

This appendix chapter delves into actor-critic methods, which combine value-based and policy-based approaches to enhance stability and performance. It introduces the Advantage Actor-Critic (A2C) algorithm, explaining how parallel workers improve exploration and reduce correlation in training data. Furthermore, it examines Asynchronous Advantage Actor-Critic (A3C), which removes synchronization constraints to enable more efficient training. A3C allows multiple agents to update the global network asynchronously, reducing idle time and improving convergence speed. These methods are crucial for tackling large-scale reinforcement learning problems with continuous state and action spaces, making them effective for complex decision-making tasks.

C.1 Advantage Actor-Critic

The A2C algorithm is summarized in Algorithm 8. Unlike methods that rely on full episode trajectories, A2C updates all states based on a fixed number

of transitions, denoted as n. The terminal state in an episode uses only the final reward and its corresponding value, whereas the first state incorporates the next n steps to compute the return. Despite the fact that the discount factor γ may seem inconsequential in practice, it significantly influences the algorithm's performance.

A2C operates in an online learning fashion, meaning that the policy is updated immediately after a few transitions rather than waiting for the entire episode to end. This characteristic makes it well-suited for problems with an infinite time horizon. Similar to value-based methods, the neural networks used for the actor and critic may suffer from correlated input data within minibatches, as consecutive states exhibit strong dependencies. However, unlike DQN, A2C does not require a replay buffer. Instead, it employs multiple independent parallel actors, ensuring better exploration and reducing correlation between sampled states. This concept, originally introduced in Asynchronous Advantage Actor-Critic (A3C), is also applicable to A2C.

Multiple parallel workers (referred to as actor-learners) are instantiated, each interacting with its own instance of the environment. At the beginning of each episode, workers receive the latest actor and critic weights from the global network. They then independently execute episodes using different seeds, ensuring that their trajectories remain uncorrelated. After completing an episode, each worker accumulates the local gradients of the actor and critic networks and sends them to the global network. The global network subsequently updates its parameters and redistributes the new weights to all workers for the next iteration. This process continues until convergence is achieved, as outlined in Algorithm 8.

Algorithm 8 Advantage Actor-Critic (A2C) Algorithm for Episodic Tasks

Input: n-step size, total episode duration T, learning rate η
Initialize: Actor network π_θ and critic network V_ϕ with random weights
Observe initial state s_0
for $t = 0, \ldots, T$ **do**
    Initialize an empty minibatch
    for $k = 0, \ldots, n$ **do**
        Select action $a_k \sim \pi_\theta(s_k)$
        Receive next state s_{k+1} and reward r_k
        Store transition (s_k, a_k, r_k, s_{k+1}) in minibatch
    end for
    If s_n is not terminal, set $R = V_\phi(s_n)$; otherwise, set $R = 0$
    Initialize gradients $d\theta \leftarrow 0$, $d\phi \leftarrow 0$
    for $k = n-1, \ldots, 0$ **do**
        Update discounted sum of rewards: $R \leftarrow r_k + \gamma R$
        Accumulate policy gradient:

$$d\theta \leftarrow d\theta + \nabla_\theta \log \pi_\theta(s_k, a_k)(R - V_\phi(s_k)) \tag{C.1}$$

        Accumulate critic gradient:

$$d\phi \leftarrow d\phi + \nabla_\phi (R - V_\phi(s_k))^2 \tag{C.2}$$

    end for
    Update actor and critic using SGD:

$$\theta \leftarrow \theta + \eta d\theta, \quad \phi \leftarrow \phi - \eta d\phi \tag{C.3}$$

end for

C.2 Parallel Advantage Actor-Critic (Parallel A2C)

The parallel version of A2C further enhances the training process by leveraging multiple workers running in parallel, as described in Algorithm 9. Each worker interacts independently with its own environment instance, eliminating correlations between sampled states. The key advantage of this approach

is that it mitigates the issue of correlated inputs, commonly encountered in reinforcement learning, by using independent trajectories generated by multiple agents.

The process begins with a global actor and critic network, whose parameters are shared across all parallel workers. Each worker copies these networks and collects an n-step trajectory from its environment. Once an episode is completed, each worker computes its local policy and value function gradients and sends them back to the global network. The global network aggregates these gradients and updates its parameters, which are then redistributed to all workers. This process continues iteratively until the model converges.

Algorithm 9 Parallel Advantage Actor-Critic (Parallel A2C)

Initialize: Global actor network π_θ and critic V_ϕ
repeatworker i in parallel
    Copy global actor π_θ and critic V_ϕ
    Sample an episode of n steps
    Compute accumulated gradients $d\theta_i$ and $d\phi_i$
    Send gradients to the global network

    Wait for all workers to complete their updates
    Merge accumulated gradients $d\theta$ and $d\phi$
    Update global actor and critic networks
until Satisfactory performance achieved

The parallel A2C framework has been effectively used in distributed reinforcement learning problems. For instance, Heydari et al. [**?**] applied this method to optimize resource allocation policies for non-cooperative mobile devices. The ability of A2C to handle large state and action spaces makes it particularly suitable for complex network optimization problems.

C.3 Asynchronous Advantage Actor-Critic (A3C)

A3C extends the parallel A2C framework by eliminating the need for synchronous updates across workers. In traditional parallel A2C, workers must complete their episodes before the accumulated gradients are merged and applied to the global network. This synchronization step can introduce inefficiencies, as faster workers must wait for the slower ones to finish. A3C overcomes this issue by allowing workers to asynchronously update the global networks in a staggered manner.

Unlike A2C, A3C does not require workers to wait before sending updates. Each worker independently interacts with the environment, collects a sequence of transitions, computes its own gradients, and immediately applies them to the global model. This results in more frequent updates and reduces idle time, making training significantly faster. The approach, summarized in Algorithm 10, leverages a technique known as HogWild! updating [**?**], which allows asynchronous gradient updates without locks, ensuring efficient parallelization.

Algorithm 10 Asynchronous Advantage Actor-Critic (A3C)

Initialize: Global actor π_θ and critic V_ϕ worker i in parallel
repeat
    Copy local version of the global networks π_θ and V_ϕ
    Sample a trajectory of n transitions
    Compute accumulated gradients $d\theta_i$ and $d\phi_i$
    Update global actor and critic networks asynchronously
until Satisfactory performance achieved

A3C has been demonstrated to achieve superior performance over A2C in complex environments.

Bibliography

[1] M. Index, "Wi-fi modulation and coding scheme (mcs) index," 2025. Accessed: February 28, 2025.

[2] R. Sutton and A. Barto, "Reinforcement learning: An introduction," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 1054–1054, 1998.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[4] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, pp. 229–256, 1992.

[5] D. T. Hoang, N. V. Huynh, D. N. Nguyen, E. Hossain, and D. Niyato, *Deep Reinforcement Learning for Wireless Communications and Networking: Theory, Applications and Implementation.* John Wiley & Sons, Inc., 2023.

[6] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," 2018.

[7] S.-i. Amari, "Natural gradient works efficiently in learning," *Neural Computation*, vol. 10, no. 2, pp. 251–276, 1998.

[8] S. M. Kakade, "A natural policy gradient," in *Advances in Neural Information Processing Systems* (T. Dietterich, S. Becker, and Z. Ghahramani, eds.), vol. 14, MIT Press, 2001.

[9] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients," *Neural Networks*, vol. 21, no. 4, pp. 682–697, 2008. Robotics and Neuroscience.

[10] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," 2017.

[11] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[12] R. Pascanu and Y. Bengio, "Revisiting natural gradient for deep networks," 2014.

[13] S. M. Kakade and J. Langford, "Approximately optimal approximate reinforcement learning," in *International Conference on Machine Learning*, 2002.

[14] S. Sharifi, S. Shahbazpanahi, and M. Dong, "A pomdp-based antenna selection for massive mimo communication," *IEEE Transactions on Com-*

munications, vol. 70, no. 3, pp. 2025–2041, 2022. This reference was used in structuring Chapter 3.

[15] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Communications Surveys Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019. This reference was used in structuring Chapter 3.

[16] O. Iacoboaiea, J. Krolikowski, Z. Ben Houidi, and D. Rossi, "Real-time channel management in wlans: Deep reinforcement learning versus heuristics," in *2021 IFIP Networking Conference (IFIP Networking)*, pp. 1–9, 2021. This reference was used in structuring Chapter 3.

[17] R. Shafin, H. Chen, Y. H. Nam, S. Hur, J. Park, Jianzhong, Zhang, J. Reed, and L. Liu, "Self-tuning sectorization: Deep reinforcement learning meets broadcast beam optimization," 2019. This reference was used in structuring Chapter 3.

[18] Y. Li, X. Zhao, and H. Liang, "Throughput maximization by deep reinforcement learning with energy cooperation for renewable ultra-dense iot networks," *IEEE Internet of Things Journal*, vol. 7, no. 9, pp. 9091–9102, 2020. This reference was used in structuring Chapter 3.

[19] J. Joung, "Machine learning-based antenna selection in wireless communications," *IEEE Communications Letters*, vol. 20, no. 11, pp. 2241–2244, 2016. This reference was used in structuring Chapter 3.

[20] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, Mar. 2016.

[21] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2016.

[22] L. C. Baird, III, "Advantage updating," *WRIGHT LAB WRIGHT-PATTERSON AFB OH*, 1993.

[23] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," 2016.