VIETNAM NATIONAL UNIVERSITY OF HOCHIMINH CITY
THE INTERNATIONAL UNIVERSITY
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

# REINFORCEMENT LEARNING - BASED BITRATE ADAPTATION FOR DYNAMIC ADAPTIVE STREAMING OVER HTTP

By
LUU MINH LONG

A thesis submitted to the School of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
Bachelor of Science in Computer Science

Ho Chi Minh City, Vietnam
2022

# REINFORCEMENT LEARNING - BASED BITRATE ADAPTATION FOR DYNAMIC ADAPTIVE STREAMING OVER HTTP

APPROVED BY:

____________________,
Vo Thi Luu Phuong, Assoc. Prof., PhD

____________________,
Le Thanh Son, MSc

____________________,
Huynh Kha Tu, MSc

____________________,
Tran Thanh Tung, PhD

____________________,
Le Duy Tan, PhD

THESIS COMMITTEE 2

# Acknowledgement

Four years ago marked my first step in Computer Science as a gaming nerd, and little did I anticipate such a ride! This path opens me opportunities that I never dreamt of, ups and downs that I must learn to adapt. As I was close to finishing my undergraduate study at International University (IU), a new journey awaited me in the future: a journey of never ending learning. These years of joyful experience would not be filled without the help of many people.

First and foremost, I would like to thank my advisor at IU, Dr. Phuong Vo. She accepted to be my advisor despite not knowing about me at all during my advisor search period. She agreed to let me choose whatever topic I wanted and whatever tool I proposed. Because of her, I learned a lot of useful knowledge that helped me a lot on my journey.

I would like to thank some people who partially supported me with the code for the this thesis, specifically: Anssi Kanervisto at University of Eastern Finland, Antonin Raffin at German Aerospace Center (DLR), Costa Huang at Drexel University, James MacGlashan at Sony AI, Nghia Nguyen, Minh Pham and my advisor Dr. Phuong Vo at IU and many more people in the RL Discord server. Without their help, the code for this thesis would not be completed.

Perhaps the people who influence my research mindset the most are the people at Carnegie Mellon University. I am in deep gratitude to Haohan Wang, who I simply met randomly on the internet in early 2021. Even I mentioned that I did not have any research experience, he accepted to teach me and kindly introduced me to his colleagues. I also thank his friend, Zeyi Huang, for spending multiple calls that lasted for hours at midnight to discuss about our project's code and possible solutions when we were stuck. Moreover, I would like to thank many people at CMU, HKUST, WISC and MBZUAI who helped to review and improve the papers, as well as provide computing resources, specifically: Zechun Liu, Zhiqiang Shen, Dong Huang, Yong Jae Lee and Eric P. Xing. Without them, two papers of mine would not have been finished. I wish to meet all of them in person after Covid, hopefully at a conference!

This thesis's writing would not be completed without the help of many people. I thank my friends: Minh Pham at International University, Khang Tran at RMIT University Vietnam, Khoi Nguyen at Vin University, Haohan Wang and Zeyi Huang at CMU, and lastly, my advisor Dr. Phuong Vo, who all provide constructive feedbacks in improving it.

Finally, this thesis is dedicated to my family: father, mother, brother, all of my relatives and friends. Thank you for being my world.

# Contents

# List of Figures

# List of Tables

# Abstract

Video Streaming is the predominant application of today's internet, contributing to over 60% of the internet's traffic. Nowadays, the video streaming market is valued at tens of billions of dollars. Concurrent to this growth is the increasing demand for the video's quality, which has shown to be one of the most important factors that directly affect the user's quality of experience. A user can abandon the watching session if there are issues related to the streaming service, such as low quality or video rebuffering, which in turn cause a loss in revenue for content providers. To tackle this problem, Adaptive Bitrate (ABR) Algorithms are widely deployed, which run on the client-side and dynamically request the quality level to the server. Traditionally, the algorithms use human-designed objectives, which take a lot of time to engineer. In recent years, Reinforcement Learning-based approaches are emerging as competitive alternatives to traditional approaches. This end-to-end solution learns to improve the streaming experience using only raw inputs such as network traces and video sizes and directly outputs the quality level, which greatly simplifies the overall process. Motivated by these works, in this thesis, I train some state-of-the-art Reinforcement Learning agents as ABR algorithms in a simulation environment using real-world video datasets and 4G LTE network traces. These algorithms are intensively tuned for 900 CPU hours to find the best set of hyperparameters. Then, the performance of the algorithms is evaluated following well-known evaluation protocols. Finally, I propose some future research directions of this field, which I believe will benefit the community as a whole.

# Chapter 1

# Introduction

The first chapter introduces the background and development history of Deep Learning, Reinforcement Learning and Video Streaming. Then, I introduce the main focus of this thesis: how Deep Learning and Reinforcement Learning can be utilized to improve Video Streaming.

## 1.1 Background

### 1.1.1 Deep Learning

Deep Learning (DL) is a branch of Artificial Intelligence (AI) that can be dated back to the XIX century[1]. DL is completely based on deep neural networks (NN) to propose an end-to-end solution by using Deep NN to automatically model the complex relationship directly between the inputs and the outputs. Nowadays, DL is being widely adopted to solve practical problems, such as image classification (Cireşan et al., 2012; Krizhevsky et al., 2012), neural machine translation (Bahdanau et al., 2016; Luong et al., 2015), and even robotics (Brockman et al., 2016).

The very first industry application of DL was proposed in 1998 when researchers invented LeNet-5: a Convolutional Neural Network (CNN) to learn and classify handwritten digits (LeCun et al., 1998). However, it was not until 2012 that DanNet (Cireşan et al., 2012) and AlexNet (Krizhevsky et al., 2012) "triggered" the DL revolution. Researchers applied deep CNN to solve million-image classification challenges and achieved human-level performance. At that moment, people realized the prospect of DL and the DL revolution began. Fast forward to today, top AI conferences such as NeurIPS, CVPR, and AAAI witnessed a breaking record of submissions in 2021 with nearly 8000 papers - more than four folds the number of submissions in 2012 (Xin, 2021). It is undoubtedly that DL is a booming research field - economists even consider it as the "fuel" of the fourth industrial revolution (Schwab, 2017).

While traditional Machine Learning approaches rely heavily on feature engineering, DL alleviates this process by transforming raw features into more complex (or so-called "deep") features Goodfellow et al. (2016). Figure 1.1 illustrates how Deep Learning differs from classical methods.

Most algorithms can belong to one of three following groups: Supervised Learning, Unsupervised Learning, and Reinforcement Learning (RL). In Supervised Learning, the NN predicts an output/label given input (e.g. classifying an image like illustrated in

[1]The paper *On the Origin of Deep Learning* (Wang et al., 2017) provides a detailed overview of the history.

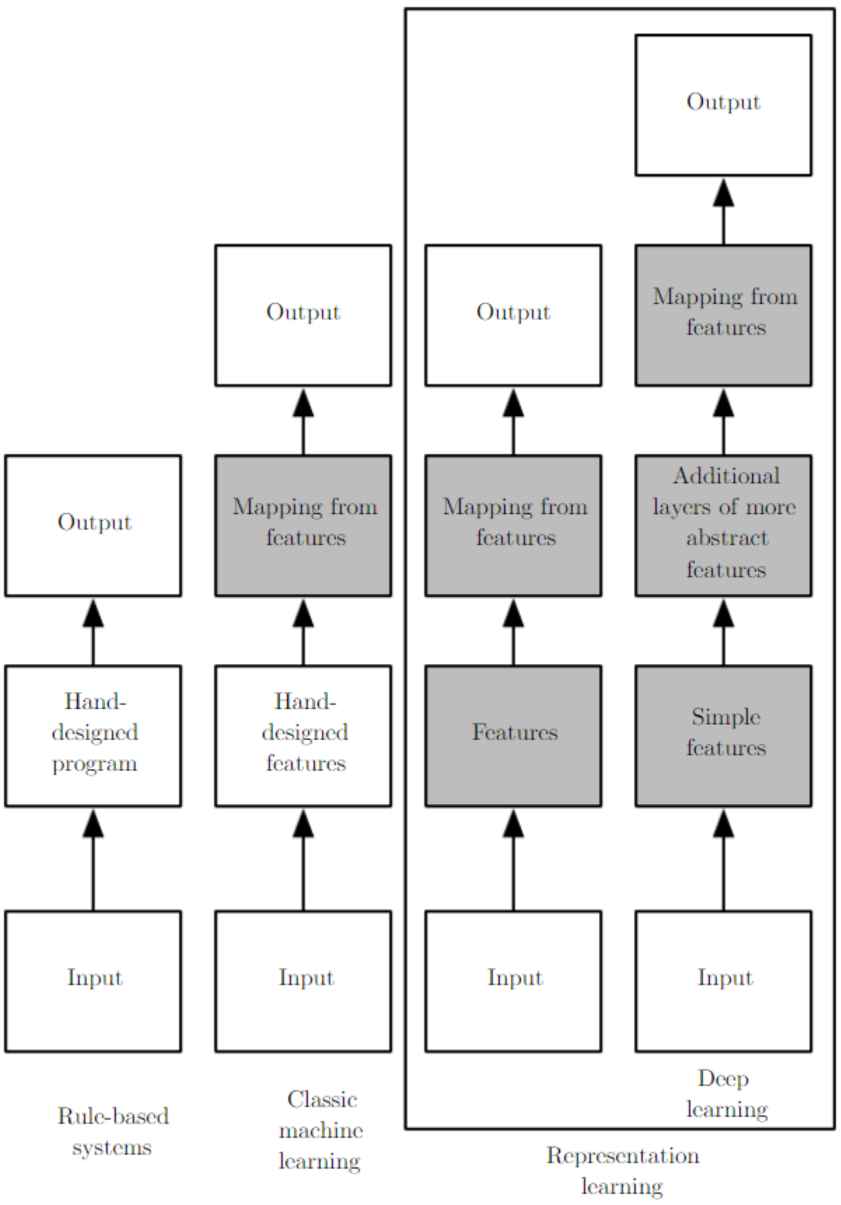


Figure 1.1: Flowcharts show the differences between rule-based, classic and representation learning. Shaded box means parts that the model can directly learn without human intervention. Image credits: (Goodfellow et al., 2016).

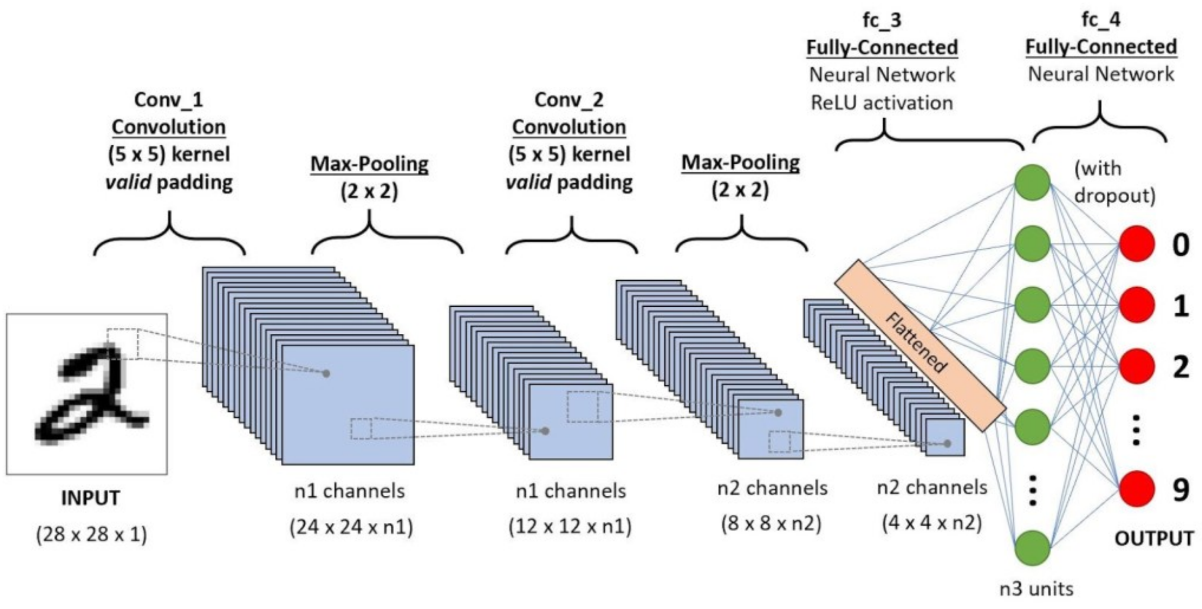


Figure 1.2: Architecture of LeNet-5 (LeCun et al., 1998), the very first application of DL. Image credits: (Barla, 2021).

Figure 1.2, predict the stock price given the price yesterday) by training on multiple pairs of input-output. In Unsupervised Learning, the NN only has inputs and is asked to learn useful properties of the structure of the data (e.g., cluster customers based on their weights and hobbies, create similar images based on the provided images). Reinforcement learning, which is introduced in the next section, is the main focus of this thesis.

### 1.1.2 Reinforcement Learning

Like DL, Reinforcement Learning (RL) also has a long history and can be dated to the 1950s. It is not until the 1980s that RL "combined" multiple threads to produce the modern field of RL. One of such threads is "optimal control", which stresses on optimizing a controller to measure a behaviour of a dynamical system over time using Dynamic Programming[2].

While supervised learning focuses on learning from labelled examples and unsupervised learning aims to learn from unlabelled ones, RL combines the best of both world. In supervised fashion, an RL learner (or so-called "agent") derives the "best" sequence of actions according to some user-defined measure (or so-called "reward signal") from interactions with an environment. In unsupervised fashion, the agent does not have a prior data on the best action for each environment state, but must learn that through trials and errors. This combination is what makes RL extremely promising: the agent can learn directly from the interactions without any prior knowledge, which is the case in most real-world problems when we do not have a dataset or the data is hard to collect. Figure 1.3 shows the interaction of an agent in an environment.

Early RL algorithms are tabular methods or dynamic programming methods - that is, the action is small and can be represented by a table. This assumption is not likely to hold true in the scenario where state-space is intractable to store in-memory, such as self-driving car. In recent years, RL is combined with DL to yield astonishing results. RL algorithms

[2]The work by (Bryson, 1996) provides an comprehensive history of optimal control.

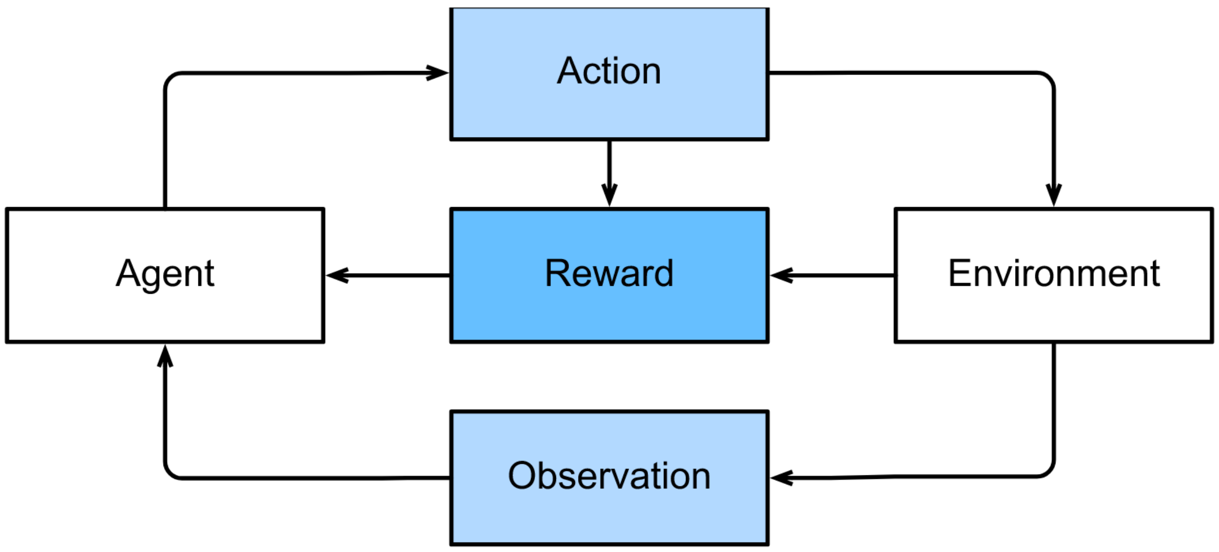


Figure 1.3: In Reinforcement Learning, an agent interacts with an environment by receiving the observation, performs an action, then receives the corresponding reward. Image credits: (Zhang et al., 2021).

have fueled many of the most publicized achievements in modern machine learning, such as Deep Q-Learning (DQN). Introduced by (Mnih et al., 2013) in 2016, DQN is the first widely successful algorithm for Deep Reinforcement Learning, which achieved human-level playing in the Atari Arcade game benchmark (Bellemare et al., 2013). Following DQN, many Deep RL algorithms have been developed: Asynchronous Advantage Actor-Critic (A3C) (Mnih et al., 2016), Trust-Region Policy Optimization (TRPO) (Schulman et al., 2015), Proximal Policy Optimization (PPO) (Schulman et al., 2017) which established the state-of-the-art performance in Atari Arcade game, MuJoCo physics simulation (Todorov et al., 2012) and OpenAI Gym (Brockman et al., 2016). These Deep RL algorithms use Deep NN instead of tabular methods, which helps RL to overcome the infinite state space problem.

In addition to games and classical controls, RL is also applied in various areas. Works like AutoAugment (Cubuk et al., 2019) applied RL to find the best image augmentation policy, (Baker et al., 2017; Zoph and Le, 2017) introduced Neural Architecture Search (NAS) to find the best NN architecture using RL, AutoMixup (Luu et al., 2021) and RL-Mix (Luu et al., 2022) utilized RL to find the suitable mix-up policies. In Video Streaming, (Gadaleta et al., 2017; Mao et al., 2017) applied DQN and A3C respectively to improve Video Streaming quality directly from the raw inputs, which is the main focus of this thesis. The next section describes the Video Streaming problem and recent advances in applying RL to improve users' experience.

### 1.1.3 Video Streaming

Streaming media is multimedia that is directly delivered and consumed in a continuous manner from a source, without knowing storage requirement in the destination. Streaming refers to the delivery method of content, rather than the content itself. The history of streaming can be dated to the 1890s, where music was "streamed" over the telephones (Reason, 2020). In 2020, the streaming market was valued at \$50.11B and its annual growth rate is anticipated to be 21% from 2021 to 2028 (GrandViewResearch, 2020). Tech giants, such as Facebook, Twitter, and YouTube are heavily investing and competing with

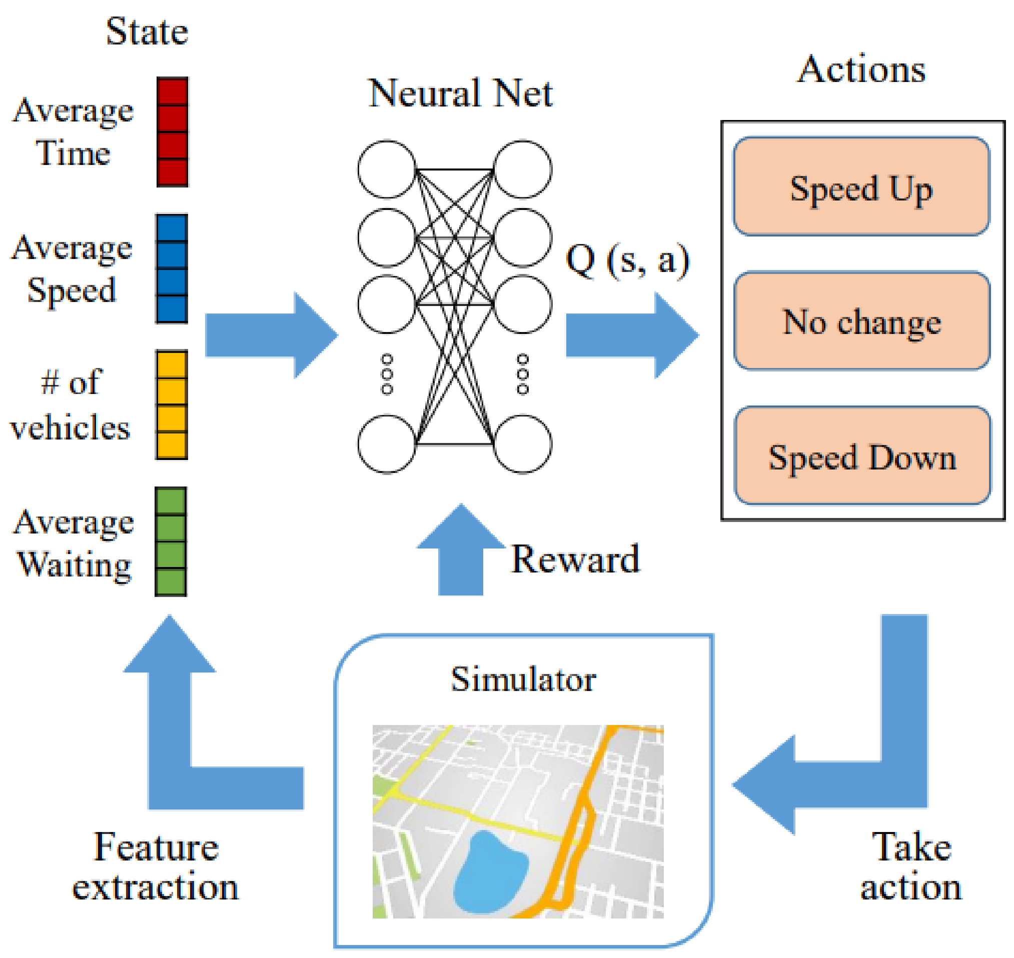


Figure 1.4: A Reinforcement Learning agent learns how to drive a car. In the simulator, the agent observes some inputs, extracts some features, then decides the actions to take. In return, the agent receives a reward. Image credits: (Tang et al., 2019).

each other to get a share of this gigantic pie (Harper, 2021).

Video streaming is the major application of today's internet, contributing to over 60% of the internet's traffic (Sandvine, 2018). Concurrent with this growth is the increasing demand for video quality. Users will abandon watching the video if there are issues related to the video quality (e.g. continuous pauses and abrupt change in video's quality), which leads to a significant revenue loss for content providers (Krishnan and Sitaraman, 2012; Dobrian et al., 2011). To tackle this problem, content providers deploy Adaptive Bitrate (ABR) algorithms, which aim to improve the quality of Experience (QoE) of the users under various circumstances to dynamically choose quality for each video *chunk* (4-second block) based on available observations such as network speed and the buffer occupancy (Figure 1.5). There are many functions defining the QoE metric, but mainly they consist of two important factors: the quality of the chunks and the amount of time that the video freezes due to rebuffering.

Selecting the bitrate for the optimal QoE is an extremely challenging task since there are underlying issues that ABR algorithms must address: (1) The variability of the network's throughput (Huang et al., 2012; Zou et al., 2015), (2) The conflict in measuring

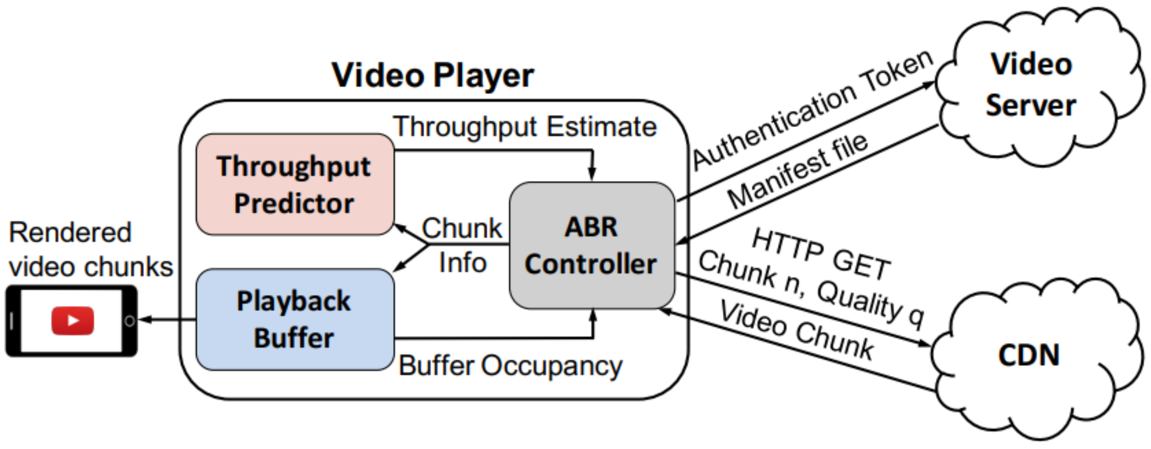


Figure 1.5: An overview of ABR algorithm's process in DASH. Image credits: (Mao et al., 2017).

QoE metric (e.g. high bitrate but minimal rebuffering), and (3) The long-time horizon (i.e. videos are encoded into several chunks, and the ABR algorithms must maximize QoE metric for all that chunks). While (Yin et al., 2015) introduces a metric that allows rebuffering, the metric proposed by (Huang et al., 2019a) introduces a harder constraint: the video must **not** be rebuffered, and it is proven that the proposed problem is a generalization of the NP-Hard Knapsack problem (Huang et al., 2019a; Kellerer et al., 2004). Even though there is no consensus on the importance of video interruptions, it is generally treated as the most disruptive experience and penalized the most in quality calculations (Huang et al., 2019a).

Proposing better ABR algorithms is a trending field of research. In recent years, many algorithms have been proposed, from throughput-based, buffer-based to Deep Learning solutions (Stockhammer, 2011; Mao et al., 2017; Spiteri et al., 2020; Yadav et al., 2017; Yin et al., 2015; Gadaleta et al., 2017). Usually, the ABR algorithms are benchmarked on DASH (Akamai, 2016) (which stands for Dynamic Adaptive Streaming over HTTP), a major technique to stream a video from the server to the user.

## 1.2 Problem Statement

A video is divided into several chunks of equal time segment (e.g.a 240-second video can be divided into 60 chunks, and each chunk stores 4 seconds of the video). The chunk is further encoded into multiple quality levels that are distributed among individual streams, which readers might find the levels' names familiar: 720p, 1080p, 1080p@30fps. For the same chunk index, a higher quality level indicates a bigger chunk size. One by one, the client requests a chunk from the server, and the chunk is downloaded to the client. Downloaded chunks are played back to the users. Figure 1.6 best illustrates the encoding and decision process.

Instead of writing the chunk directly to the storage of the client, the downloaded chunks are stored in the "Replay Buffer" before being played, which in turn is stored in RAM (Random Access Memory). The replay buffer size is pre-determined depending on the client, but usually, it can hold tens of seconds. The replay buffer uses the first-in-first-out rule: chunks that are downloaded first will be played first. The chunk cannot be

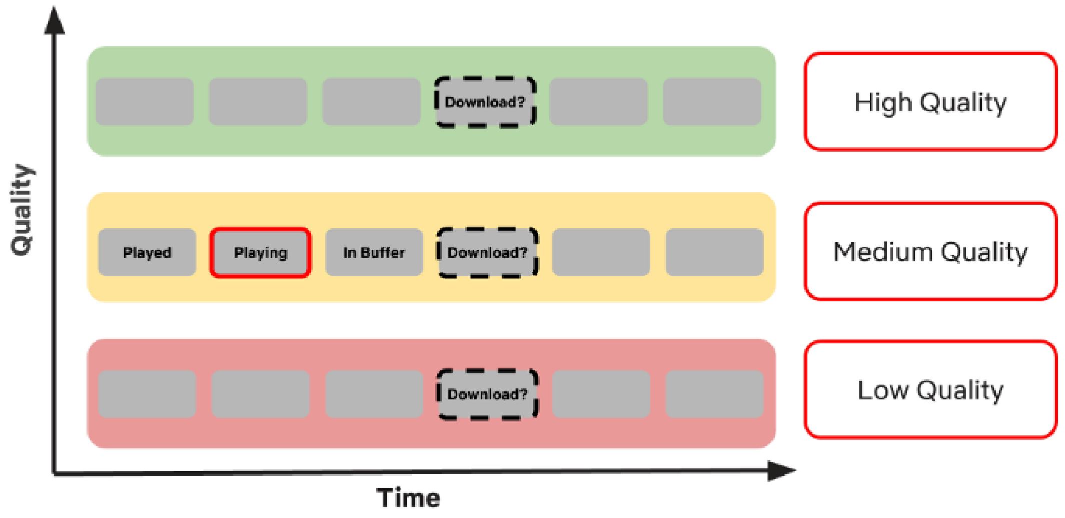


Figure 1.6: Visualization of the video streaming decision. A chunk can be either played, playing, or in the buffer. The ABR algorithm decides which chunk quality to download for the future chunks that are not in the replay buffer. Image credits: (Ekanadham, 2018).

played until it is fully downloaded to the replay buffer.

The video player requires an initial replay buffer level before starting to play, i.e. Several chunks are pre-downloaded and will not be used in the optimization objective. When the replay buffer exceeds an upper threshold level, the video player will stop requesting new chunks. The client must wait for the buffer level to decrease below the upper threshold to send the request for a new chunk again. The video player *rebuffers* (freezes) when the chunk index which is going to be played next does not exist in the buffer. The video player then pauses playing and waits for the new chunk until it is completely downloaded.

Adaptive Bitrate (ABR) algorithms aim to optimize the Quality of Experience (QoE) of the users under various circumstances to dynamically choose quality for each video chunk based on available observations such as estimated network throughput and the replay buffer. To measure the quality of the ABR algorithm, in this work we focus on two common functions: one proposed by (Yin et al., 2015) and one proposed by (Huang et al., 2019a). In general, both functions have the same two main contributing factors: the quality of the chunk, and the amount of time the video freezes.

(Yin et al., 2015) proposes a QoE metric (which hereby I refer it as **I-QoE** (interruption-allowed QoE)) that maximizes the sum of quality, allows video interruption but penalties the overall QoE if (1) the video is interrupted, (2) if the quality levels of two consecutive chunks switch. Furthermore, it limits the replay buffer size to a fixed threshold. In this work, I focus on this metric, since it is more popular in later works (Mao et al., 2017; Spiteri et al., 2020), and can be evaluated at scale (Huang et al., 2019a).

The metric proposed by (Huang et al., 2019a) (which hereby I refer it as **NI-QoE** (non-interrupted QoE)) can be formalized as the maximum sum of quality that can be downloaded while maintaining an **uninterrupted** video playback (i.e. chunk index $n$ must finish downloading before chunk index $n-1$ finishes playing) but does not limit the replay buffer size. This problem formulation is known to be a generalization of the NP-Hard Knapsack problem (Kellerer et al., 2004). The generalization will be discussed in Section 2.3.

In this thesis, different Reinforcement Learning algorithms are compared to improve QoE for the clients. Specifically, three algorithms: DQN, A2C, and PPO are trained and intensively evaluated to determine suitable algorithms that can be deployed in DASH. Algorithms are evaluated using the I-QoE metric by (Yin et al., 2015).

## 1.3 Contribution and Scope

In this thesis, I make the following contributions:

1. Due to limited computing resources and funding, hyperparameter combinations of each algorithm are searched only for 120 trials each. Each trial takes approximately 2.5 hours on a 4 cores Intel Xeon @2.3GHz CPU. In total, the total runtime of RL algorithms is approximately 38 CPU days, or 900 hours.

2. The performance is evaluated using the I-QoE metric.

3. Only model-free Reinforcement Learning algorithms are considered. Specifically, they are DQN, A2C, and PPO.

4. Other algorithms, such as random search, throughput-based (Stockhammer, 2011) and BOLA-U (Spiteri et al., 2020) are implemented to compare.

5. The work is performed in a simulated environment that is compatible with the OpenAI Gym interface (Brockman et al., 2016), which makes it easier to be reused.

On the other hand, there are some limitations I do not cover. These aspects will be revisited in Section 5.2.

## 1.4 Structure of the Thesis

The structure of this thesis is as follows:

- Chapter 2 reviews the basic knowledge in order to follow the thesis. Specifically, section 2.1 reviews Deep Learning, section 2.2 reviews Reinforcement Learning, and section 2.3 reviews the NP-Hardness of the NI-QoE problem.

- Chapter 3 introduces the libraries used and the implementation details, as well as the decisions to use these libraries.

- Chapter 4 describes the results and evaluation protocol.

- Chapter 5 concludes the work and proposes future directions.

# Chapter 2

# Literature Review

## 2.1 Deep Learning

This section briefly introduces background knowledge about Deep Learning. Specifically, I introduce the intuition behind supervised learning, neural networks, the Universal Approximation Theorem, and give some empirical results on learning a non-linear function. Readers can refer to great materials such as (Zhang et al., 2021; Goodfellow et al., 2016; Wang et al., 2017) for a detailed overview.

### 2.1.1 Empirical Risk Minimization

To the best of my knowledge, the idea of supervised learning is pioneered by (Vapnik, 1992). The learning process is described by three components:

1. A generator of the inputs $\mathbf{X}$, which are idenpendently and identically drawn from an unknown distribution $P(x)$.

2. An output vector $y$ to every input $x$, drawn from a conditional but unknown distribution $P(y|x)$. A collection of these inputs is denoted $\mathbf{Y}$.

3. A learning model capable of learning a best-response mapping function from $X$ to $Y$, $f(x;\theta) : X \rightarrow Y$ with the parameter $\theta$.

Usually, we have a training dataset consisting of $n$ instances: $(x^{(1)}, y^{(1)}), ..., (x^{(n)}, y^{(n)})$ where $x^{(i)} \in \mathbf{X}, y^{(i)} \in \mathbf{Y}$ which we assume there exists a joint distribution $P(x, y)$ and the instances are drawn i.i.d. from it. In order to choose the best approximating function, we measure the non-negative, real-valued loss between the learning function and the outputs: $L(y, f(x;\theta))$. The associated risk is defined as the expectation of the loss function:

$$R(f) = \mathbb{E}[L(y, f(x;\theta))] = \int L(y, f(x;\theta))dP(x, y) \tag{2.1}$$

Some commonly used loss functions for regression tasks are Mean Absolute Error (MAE): $L(y, y') = \frac{1}{2}|y - y'|^2$ and Mean Squared Error (MSE): $L(y, y') = \frac{1}{2}(y - y')^2$ where $y' = f(x;\theta)$ is the prediction of the label $y'$ given the input $x$ and the parameter $\theta$.

The ultimate goal is to find a function $f^*$ such that the risk $R(f)$ is minimal:

$$f^* = \arg\min_{f \in \mathcal{F}} R(f) \tag{2.2}$$

However, the risk $R(f)$ cannot be direct calculated with zero loss because the distribution $P(x, y)$ is unknown. Generally, we are interested in computing an approximation of the risk, called *empirical risk*, by taking the mean of the loss function on the available training set:

$$R_{emp}(f) = \frac{1}{n}\sum_{i=1}^{n} L(y^{(i)}, f(x^{(i)}; \theta)) \tag{2.3}$$

The empirical risk minimization (ERM) principle (Vapnik, 1992) states that the learning algorithm should choose a hypothesis $\hat{f}$ which minimizes the empirical risk:

$$\hat{f} = \arg\min_{f \in \mathcal{F}} R_{emp}(f) \tag{2.4}$$

However, ERM for a classification problem with the 0-1 loss function is proven to be NP-Hard (Feldman et al., 2009) even for simple classifiers. Instead, we are interested in finding a hypothesis $\hat{f}$ that has low ERM error.

### 2.1.2 Maximum Likelihood Estimation

Maximum Likelihood Estimation (MLE) is a special case of ERM where the loss function is the negative log-likelihood function. In MLE, we assume that data is sampled i.i.d from a distribution that comes from a family of probability functions parameterized by $\theta$ and wishes to find $\hat{\theta}$ such that the likelihood of the sample reaches its maximum. Since each instance $(x^{(i)}, y^{(i)})$ is sampled i.i.d from a joint distribution $P(x, y; \theta)$ (stated in the previous section), it is trivial that

$$P((x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)})) = \prod_{i=1}^{n} P(x^{(i)}, y^{(i)}; \theta) \tag{2.5}$$

For ease of use, in terms of computation and machine precision, we take the logarithm of both sides to obtain the following equation:

$$\log P((x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}); \theta), \ldots, (x^{(n)}, y^{(n)})) = \sum_{i=1}^{n} \log P(x^{(i)}, y^{(i)}; \theta) \tag{2.6}$$

Since the logarithm of a function preserves its concavity, the optimal $\hat{\theta}$ is the same after the logarithm operation is performed. As a result, we obtain an optimization problem that resembles ERM using negative log-likelihood estimation:

$$\hat{\theta} = \arg\min_{\theta} \sum_{i=1}^{n} -\log P(x^{(i)}, y^{(i)}; \theta) \tag{2.7}$$

Equation (2.7) is the basis of various popular loss functions used in Machine Learning, such as mean-squared error (by assuming that features and labels are sampled from a multivariate normal distribution) and cross-entropy (by assuming that the joint distribution of features and labels is the Bernoulli distribution).

### 2.1.3 Neural Networks

#### Neuron

Before we describe a NN, it is best to know the units that make up the NN - the neuron. Neurons have learnable parameters called weights and biases. Each neuron

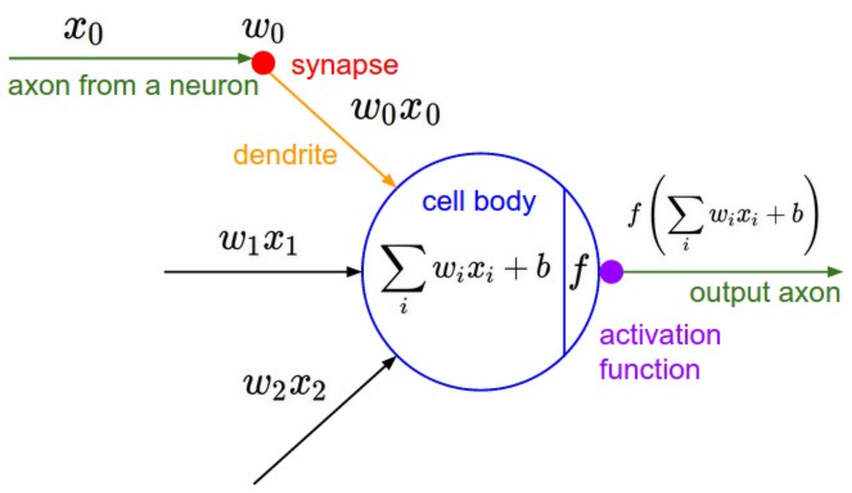


Figure 2.1: Visualization of a neuron. Here $x_i$ is the $i$-th input feature and $w_i$ is the associated weight. A neuron calculates the dot product $\mathbf{WX}$ plus a bias term, and passes the result through an activation function $\sigma$, usually is a non-linear one. Image credits: (Li et al., 2021a).

receives some inputs as numbers, performs multiplication and addition operations, and optionally follows it with a non-linear activation function. A layer consists of multiple neurons stacked vertically. A hidden layer in the NN is made up of a set of neurons, where each neuron is fully connected to all neurons in the previous layer, and where neurons in a single layer function completely independently and do not share any connections. The last layer is called the "output layer" and in classification settings, it represents the probability of the classes. Figure 2.1 shows how a neuron works.

### Deep Feedforward Networks

Deep feedforward networks, usually called Deep Neural Networks for short, are the heart of DL algorithms. Like MLE, the goal of NN is to approximate a mapping function $y = f(x;\theta)$ and learn the values of the parameters $\theta$ that result in the best function approximation, usually by minimizing a loss function, similar to the objective of ERM and MLE.

These models are called feedforward NN because the inputs flows through the function are evaluated from $x$, through the immediate computations to define $f$, and finally, give the output $y$. They are called network because typically they are represented by composing together different functions through a chain, for example: $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$ denotes a three-layer network, where $f^{(i)}$ denotes layer $i$ (Figure 2.2). The most important aspect of NN is the function of the layer (at least one) should be non-linear. The reason is stacking multiple linear layers results in a single linear layer. Consider the linear two-layer network as follows:

$$\begin{aligned} f^{(1)} &= \mathbf{X}\theta^{(\mathbf{1})} \\ f^{(2)} &= f^{(1)}\theta^{(\mathbf{2})} \end{aligned} \tag{2.8}$$

For simplicity, we opt out the bias term. Then, $f^{(2)} = f^{(1)}\theta^{(\mathbf{2})} = \mathbf{X}\theta^{(\mathbf{1})}\theta^{(\mathbf{2})} = \mathbf{X}\theta$. Thus, we can view the equivalence formally by proving that for any value of the weights, we can just collapse out the hidden layer, yielding an equivalent single-layer model with parameters $\theta$.

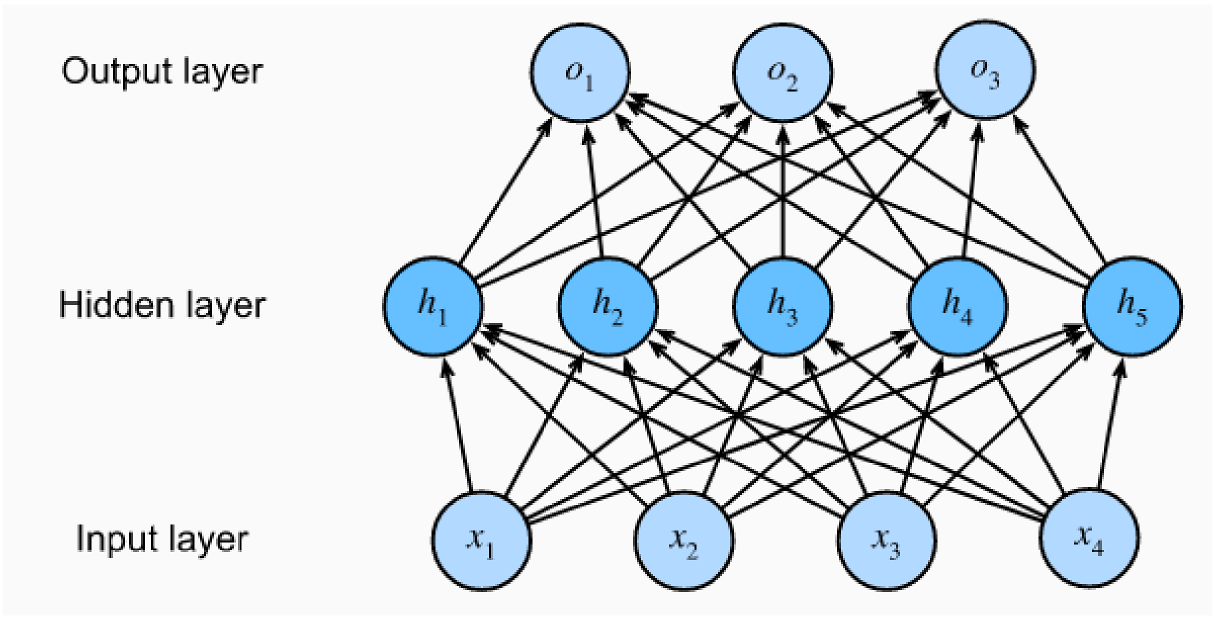


Figure 2.2: An illustration of a three-layer NN. Here, $x_i$ is the $i$-th input feature and $o_i$ is the $i$-th output features. Image credits: (Zhang et al., 2021).

To help NN learn non-linear functions, we simply plug a non-linear function $\sigma$ to each layer after the computation:

$$\begin{aligned} f^{(1)} &= \sigma(\mathbf{X}\theta^{(\mathbf{1})}) \\ f^{(2)} &= \sigma(f^{(1)}\theta^{(\mathbf{2})}) \end{aligned} \tag{2.9}$$

There are a lot of choices for the activation function, such as $\max(0, x)$, $\frac{1}{1+exp(-x)}$, which are known as ReLU and Sigmoid respectively. The NN is trained using the back-propagation algorithm (Rumelhart et al., 1987), which is skipped in this thesis due to the extensive resources available.

## 2.1.4 Universal Approximation Theorem

One remarkable property of neural networks, widely known as the universal approximation property, roughly describes that an NN can represent any continuous function with any desired accuracy. This property has three different aspects:

- Boolean Approximation: an NN of one hidden layer can calculate any Boolean function exactly.
- Continuous Approximation: an NN of one hidden layer can approximate any bounded continuous function with any desired accuracy.
- Arbitrary Approximation: an NN with two hidden layers can approximate any function with arbitrary accuracy.

The first aspect is straightforward. Based on De Morgan's laws, every boolean formula can be converted into an equivalent Conjunctive Normal Form, which consists of only OR and AND operations. Therefore, we simply rewrite the target Boolean function into an OR of multiple AND operations. Thus, we can design the network such that the input layer performs all AND operations, and the hidden layer is simply an OR operation.

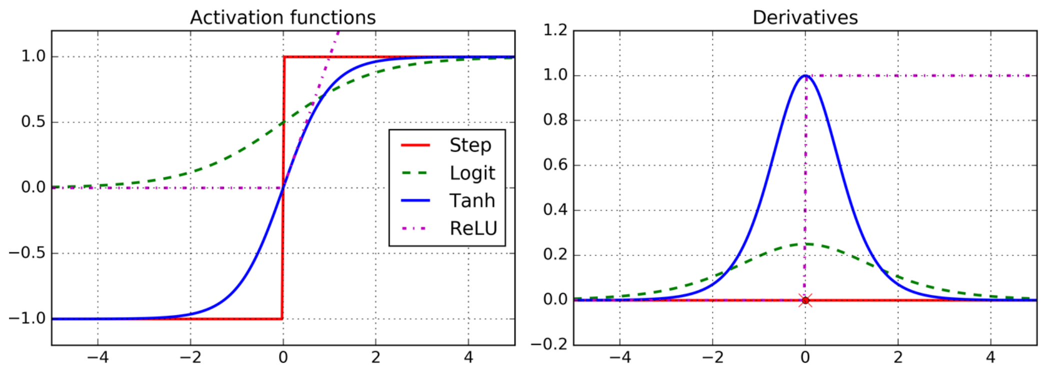


Figure 2.3: Activation Functions and their derivatives. Image credits: (Géron, 2019).

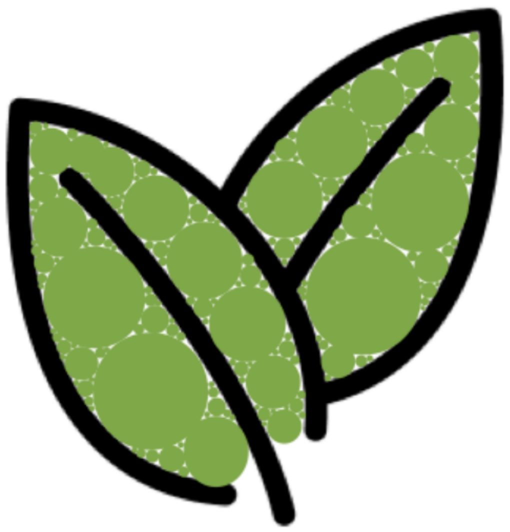


Figure 2.4: Approximating a leaf using circles. Image credits: (Wang et al., 2017).

For the second aspect, readers can refer to (Cybenko, 1989; Hornik et al., 1989). Specifically, work by (Cybenko, 1989) has shown that a one hidden layer neural network (input - one hidden - output) can approximate any bounded continuous function with arbitrary accuracy. Specifically, if we have a function $f(x)$ of the form of a two-layer network:

$$f(x) = \sum_i \theta_i^{(2)} \sigma(\theta_i^{(1)} \mathbf{X} + \mathbf{b}) \tag{2.10}$$

then $f(x)$ is dense in the subspace of where it is in. In other words, for an arbitrary function $g(x)$ in the same subspace as $f(x)$, we have $|f(x) - g(x)| < \epsilon$ where $\epsilon > 0$.

For the third aspect, readers can refer to (Hornik et al., 1989; Cybenko, 1988).

### 2.1.5 Playground: classify two circles

The following part introduces a simple NN which learns a non-linear classification problem. The experiment is conducted at the following URL:

https://playground.tensorflow.org/

We will use the two circles dataset - that is, a smaller circle inside a circle. Figure 2.5 demonstrates the configuration and the result. First, a simple NN without any activation function is used. Second, some layers are added to the NN. The visualizations show that the NNs are incapable of learning the boundary in 1000 epochs.

The story is different if a non-linear activation is used. Figure 2.6 illustrates the result of using ReLU. Even with one layer, the NN can determine the boundary in 1,000 epochs. Furthermore, when more neurons are added to the layer, the NN converges even faster - it is able to classify the two circles in just 350 epochs!

## 2.2 Reinforcement Learning

We briefly describe RL and DRL algorithms. Parts of this section are from (Sutton and Barto, 2018; Achiam, 2018), which the readers can refer to gain more insights.

### 2.2.1 General

A RL problem center parts are the agent and the environment, where the environment is the world that the agent interacts with. Every step, the agent receives an observation (generally partial) of the state of the world and then computes on an action to take. The environment alters when the action of the agent is taken on it, but may also continously changes on its own.

After the action step, the agent also receives a scalar reward from the environment telling it the outcome of the action. Ultimately, the goal of the agent is to maximize its cumulative reward over multiple episodes, called return.

### 2.2.2 States and Observations

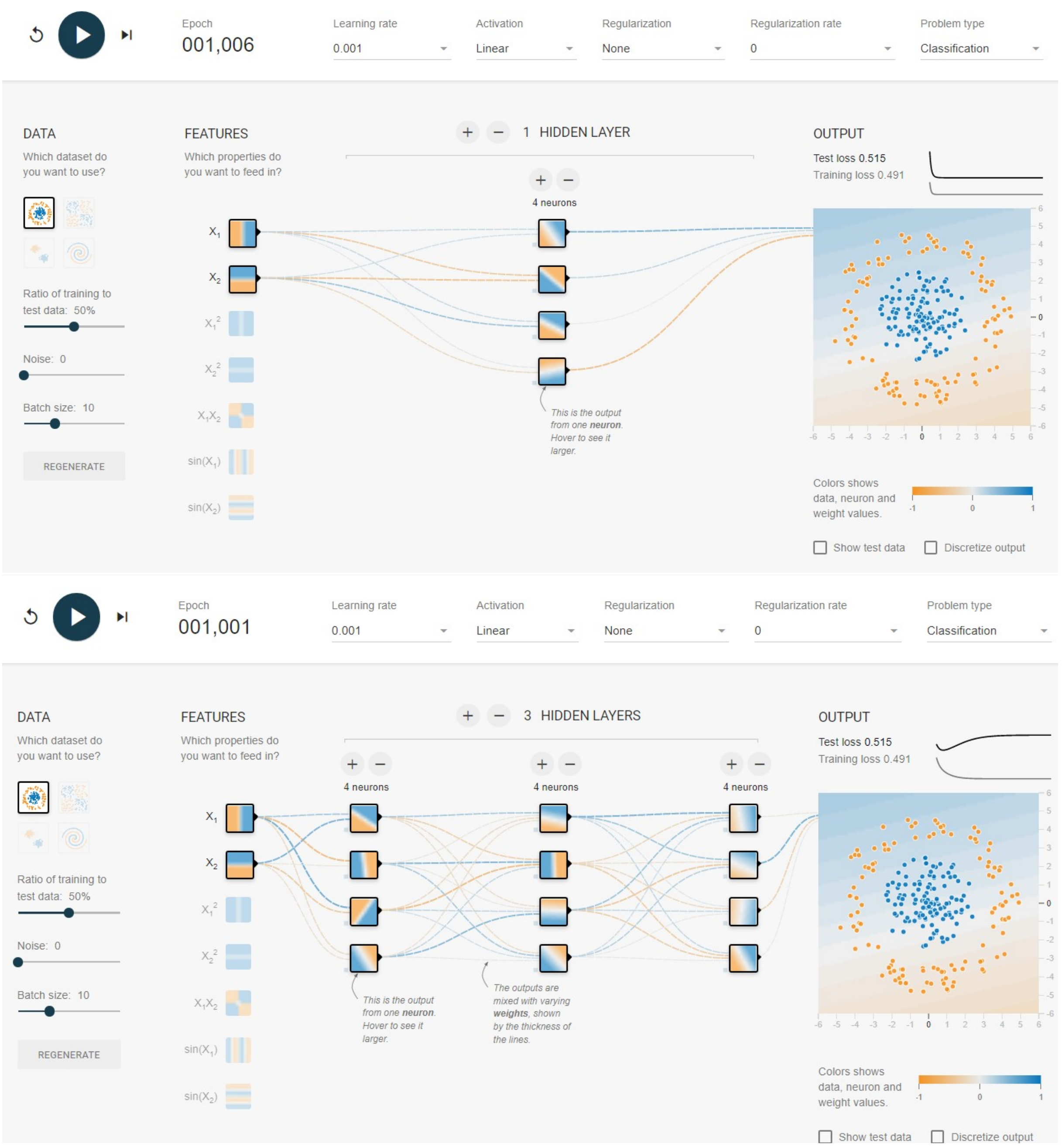
A state $s$ is a complete representation of the state of the world. There is no information that is hidden in the state. On the other hand, an observation $o$ is a partial description of a state, which may omit partial representations of the state (e.g., an image of a game).An observation can be formalized as a hidden function of representation $h$ of the state: $o = h(s)$.

In Deep RL, we almost always represent states and observations by tensors of multiple dimensions. For instance, a image observation could be represented by the RGB matrix of its pixel values.

When the agent is able to observe the state of the environment, we say that the environment is fully observed. When the agent can only see a partial observation, we say that the environment is partially observed. Depending on the partiality of the observation, further modifications might be required (e.g., RNN-based models, frame stacking, etc.)

### 2.2.3 Action space

The action space is different depending on the environment. The set of all valid actions in a given environment is often called the action space. Some environments, like Atari games and Go chess, have discrete action spaces, where only a finite number of moves are available. Other environments, like a robot in a physical world, have continuous action spaces. In continuous spaces, actions are real-valued vectors.


Epoch 001,006
Learning rate 0.001
Activation Linear
Regularization None
Regularization rate 0
Problem type Classification

Ratio of training to test data: 50%
Noise: 0
Batch size: 10

1 HIDDEN LAYER
4 neurons

Test loss 0.515
Training loss 0.491

Epoch 001,001
Learning rate 0.001
Activation Linear
Regularization None
Regularization rate 0
Problem type Classification

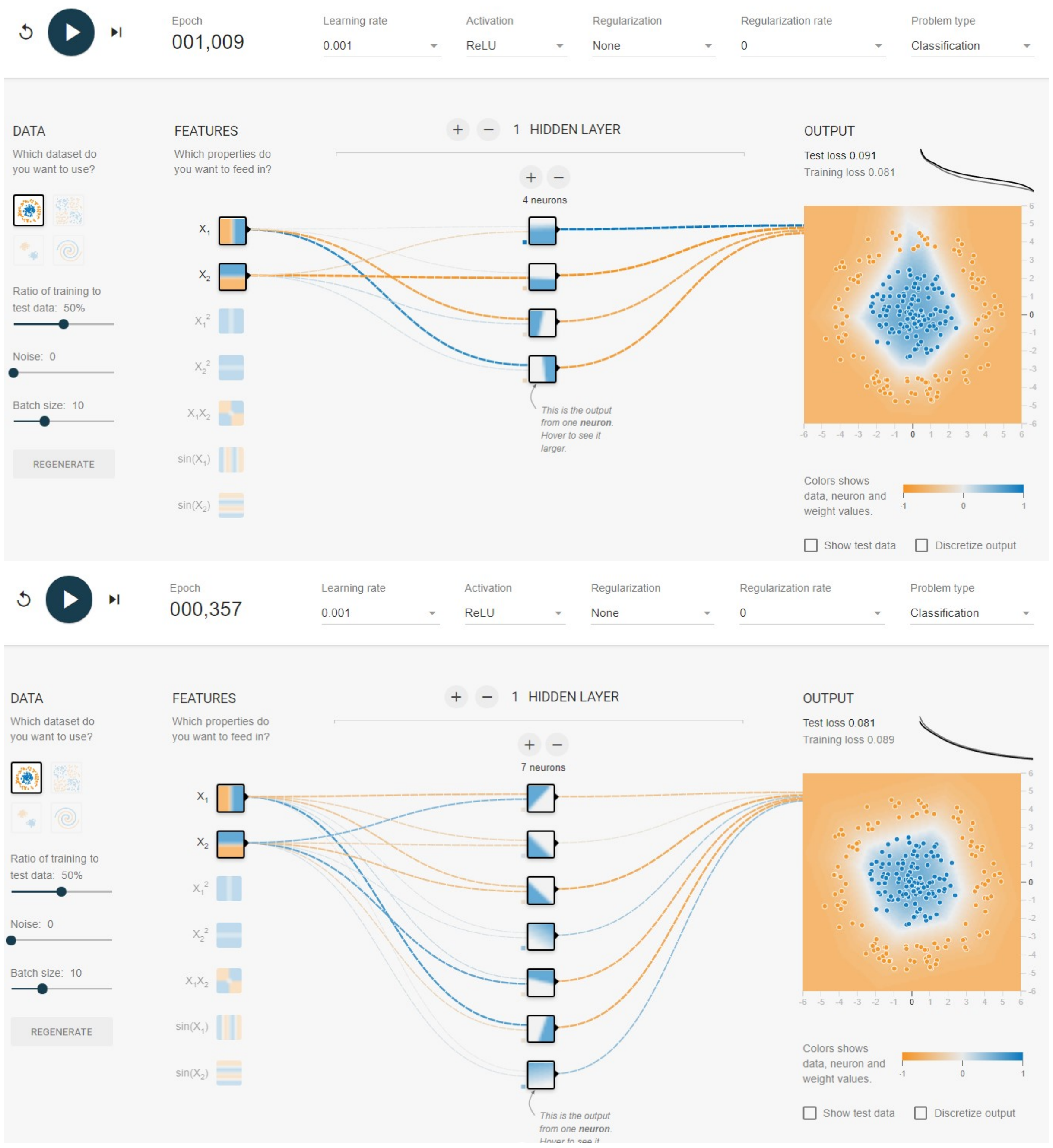
Ratio of training to test data: 50%
Noise: 0
Batch size: 10

3 HIDDEN LAYERS
4 neurons
4 neurons
4 neurons

Test loss 0.515
Training loss 0.491

DATA
Which dataset do you want to use?
REGENERATE
FEATURES
Which properties do you want to feed in?
$X_1$
$X_2$
$X_1^2$
$X_2^2$
$X_1X_2$
$\sin(X_1)$
$\sin(X_2)$
This is the output from one **neuron**. Hover to see it larger.
The outputs are mixed with varying **weights**, shown by the thickness of the lines.
OUTPUT
Colors shows data, neuron and weight values.
-1
0
1
Show test data
Discretize output

Figure 2.5: A linear NN learns to classify two circles.


Epoch 001,009 Learning rate 0.001 Activation ReLU Regularization None Regularization rate 0 Problem type Classification

DATA Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE

FEATURES Which properties do you want to feed in?

1 HIDDEN LAYER
4 neurons

This is the output from one **neuron**. Hover to see it larger.

OUTPUT
Test loss 0.091
Training loss 0.081

Colors shows data, neuron and weight values.
Show test data Discretize output

Epoch 000,357 Learning rate 0.001 Activation ReLU Regularization None Regularization rate 0 Problem type Classification

DATA Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE

FEATURES Which properties do you want to feed in?

1 HIDDEN LAYER
7 neurons

This is the output from one **neuron**. Hover to see it

OUTPUT
Test loss 0.081
Training loss 0.089

Colors shows data, neuron and weight values.
Show test data Discretize output

Figure 2.6: A non-linear NN learns to classify two circles.

The action space greatly influences the agent's performance. Recent studies (Kanervisto et al., 2020) suggest that removing all but the necessary actions required to complete the task boosts the performance by a large margin.

### 2.2.4 Policy

A policy is a "rule" of by an agent to decide what actions to take depending on the state. It can be deterministic, in which case it is denoted by $\mu$:

$$a_t = \mu(s_t) \tag{2.11}$$

or it may be stochastic, in which case it is denoted by $\pi$:

$$a_t \sim \pi(\cdot|s_t) \tag{2.12}$$

In deep RL, the policy is paramterized: policies whose outputs are learnable functions that depend on a neural network which we can adjust to change the behavior via RL optimization algorithms.

Generally, the neural network of such a policy is denoted as $\theta$:

$$\begin{aligned} a_t &= \mu_\theta(s_t) \\ a_t &\sim \pi_\theta(\cdot|s_t) \end{aligned} \tag{2.13}$$

### 2.2.5 Trajectory

A trajectory (also frequently called episodes or rollouts) $\tau$ is a sequence of states-actions-rewards in the world, which is usually assumed to follow the Markov Decision Process:

$$\tau = (s_0, a_0, s_1, a_1, ...). \tag{2.14}$$

The very first state of the world, $s_0$, is randomly sampled from the start-state distribution, sometimes denoted by $\rho_0$:

$$s_0 \sim \rho_0(\cdot) \tag{2.15}$$

Transitions of states (that is, what happens to the world between the state at time $t, s_t$, and the state at $t+1, s_{t+1}$, are controlled by the inner laws of the environment, and depend on only the most recent action, $a_t$. They can be deterministic: $s_{t+1} = f(s_t, a_t)$, or stochastic: $s_{t+1} \sim P(\cdot|s_t, a_t)$.

### 2.2.6 Reward and Return

The reward function $R$ is of paramount importance in RL. Typically, it depends on the current state of the environment, the agent's decision of the action, and the next state of the world:

$$r_t = R(s_t, a_t, s_{t+1}) \tag{2.16}$$

Although frequently this is simplified to just a dependence on the current state, $r_t = R(s_t)$, or state-action pair $r_t = R(s_t, a_t)$. The goal of the agent is to maximize the cumulative reward over a trajectory. We will denote all of these cases with $R(\tau)$.

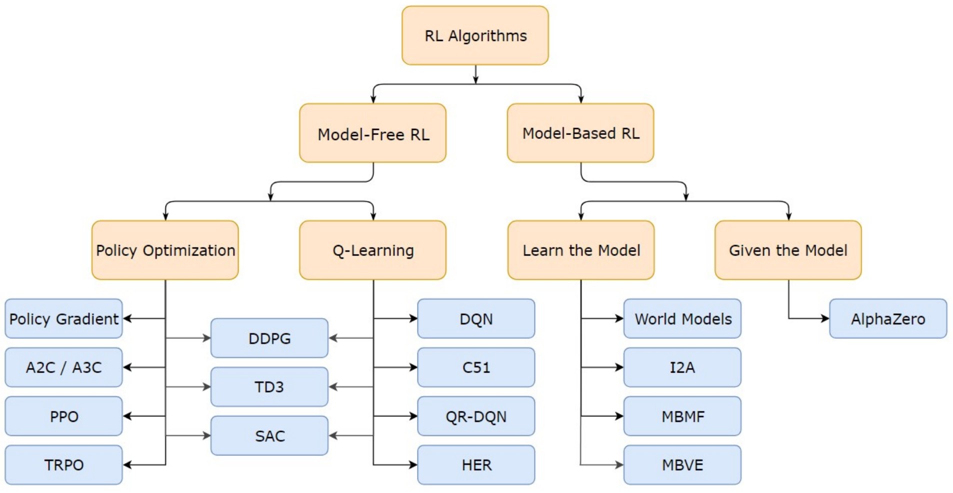


Figure 2.7: RL algorithm landscape. Image credits: (Achiam, 2018).

There are two common types of return. One kind is the finite-horizon un-discounted return, which is just the sum of rewards obtained in a defined number of steps:

$$R(\tau) = \sum_{t=0}^{T} r_t \tag{2.17}$$

Another type of return is the infinite-horizon discounted return, which is the sum of all rewards ever obtained by the agent but discounted by a discount factor $\gamma \in (0, 1)$:

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t. \tag{2.18}$$

The discount factor is both intuitively appealing and mathematically convenient. On an intuitive level: cash now is better than cash later. Mathematically: an infinite horizon sum of rewards may not converge to a finite value and is hard to deal with in equations. However, with a discount factor and under reasonable conditions, the infinite sum converges because it is a geometric sum of a series.

### 2.2.7 Q-function, V-function and the Advantage

The *Q-function* or *action-value* function of a given policy $\pi$ is defined as $Q^\pi(s_t, a_t) = \mathbb{E}_\pi[R_t | s_t, a_t]$ tells the expected return if the agent starts in state $s_t$, takes action $a_t$ and then forever acts according to $\pi$, while the *V-function* or *state-value* function is defined as $V^\pi(s_t) = \mathbb{E}_\pi[R_t | s_t]$ tells the expected return if the agent starts in state $s_t$ and always acts according to $\pi$. The value $A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$ is called the *advantage* and tells whether the action $a_t$ is better or worse than an average action the policy $\pi$ takes in the state $s_t$.

In practice, the policy and value functions are represented as neural networks. Specifically, we maintain one neural network for the policy $\pi$, and one neural network to approximate the value function of the current policy $V \approx V^\pi$.

The most used technique to compute the advantage function is Generalized Advantage Estimation (Schulman et al., 2016). The generalized formula can easily be applied to policy-gradient-based algorithms:

$$A_t(s_t, a_t) = \delta_t + (\gamma\lambda)\delta_{t+1} + ... + (\gamma\lambda)^{T-t+1}\delta_{T-1} \tag{2.19}$$

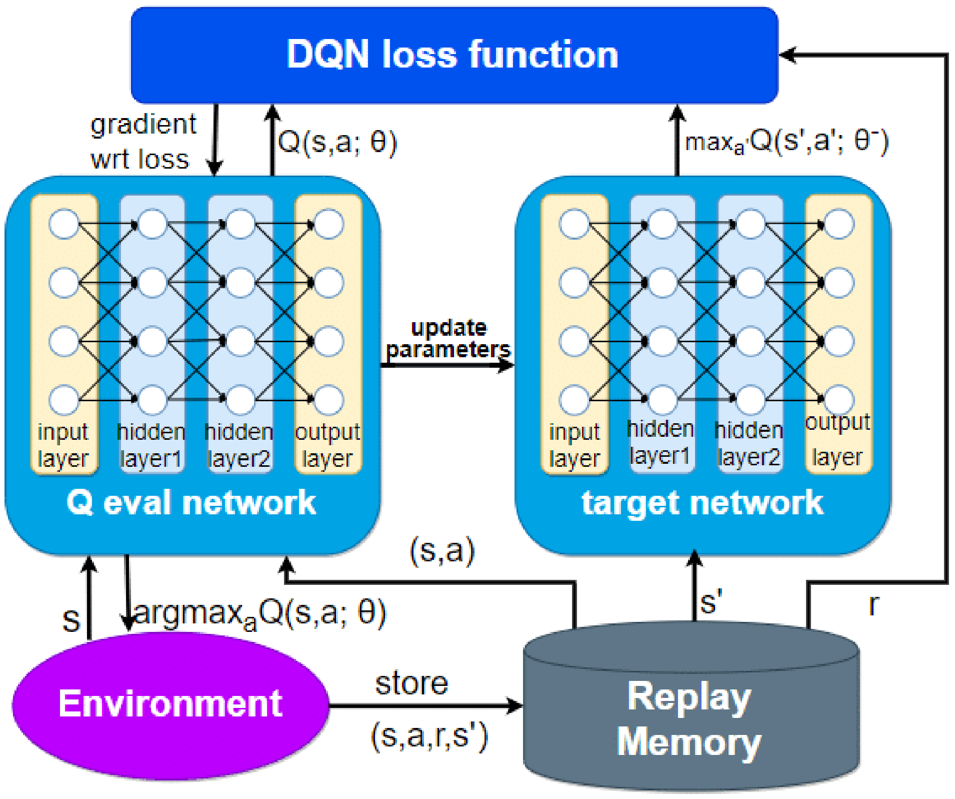


Figure 2.8: A visualization of DQN. Image credits: (Zhou et al., 2019)

where $T$ denotes the maximum length of the trajectory but not the terminal step of a complete task, and $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$ which is known as the TD-error. GAE is commonly used in Policy Gradient-based algorithms.

### 2.2.8 The RL Landscape

The first and foremost criteria to characterize an RL algorithm is whether if it is **Model-based** or **Model-free**. By the Model we mean the underlying dynamics of the transitions (e.g., whether it is possible to predict the state transitions and rewards). By "free" we mean the agent has no access or the main objective is not to learn the Model. In this work, we focus on Model-free algorithms, as they are more cost-effective in general.

### 2.2.9 Deep Q-Learning

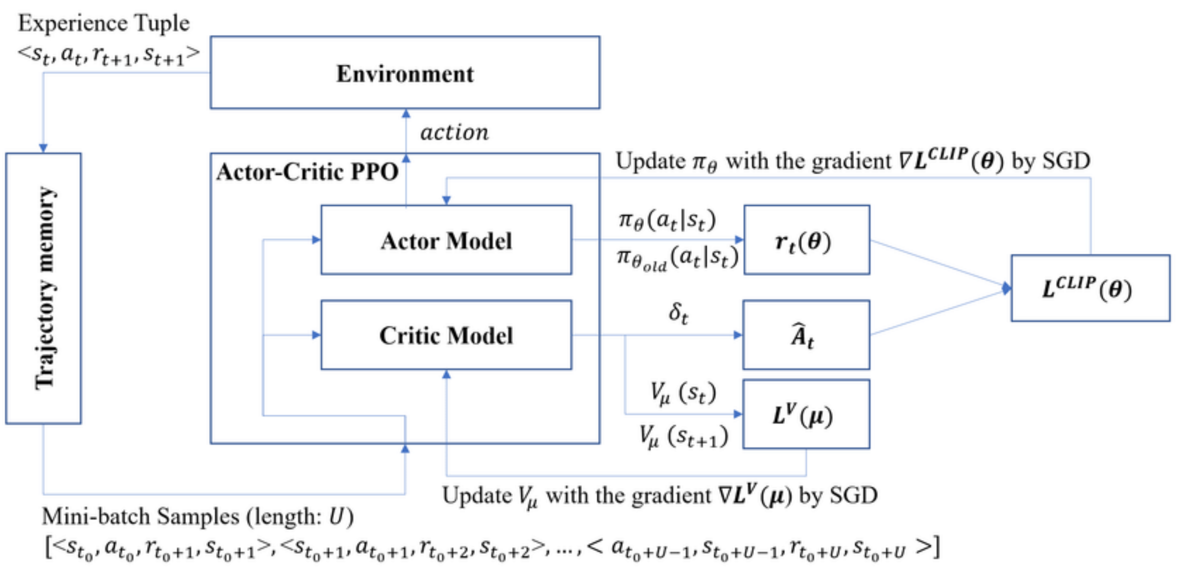
Deep Q-Learning (Mnih et al., 2013) is a type of Q-Learning which uses a neural network to approximate the optimal action-value function. If the optimal $Q^*(s,a)$ is known, then in any given state, $a^*(s)$ can be found by using:

$$a^*(s) = \arg\max_a Q^*(s,a)$$

The Bellman equation defines $Q^*(s,a)$ as:

$$Q^*(s,a) = \mathop{\mathrm{E}}_{s' \sim P}\left[r(s,a) + \gamma \max_{a'} Q^*(s',a')\right] \tag{2.20}$$

where $s' \sim P$ is shorthand for saying that the next state, $s'$, is sampled by the environment from a distribution $P(\cdot|s,a)$.



Experience Tuple
<$s_t, a_t, r_{t+1}, s_{t+1}$>
Environment
action
Trajectory memory
Actor-Critic PPO
Actor Model
Critic Model
Update $\pi_\theta$ with the gradient $\nabla L^{CLIP}(\theta)$ by SGD
$\pi_\theta(a_t|s_t)$
$\pi_{\theta_{old}}(a_t|s_t)$
$r_t(\theta)$
$L^{CLIP}(\theta)$
$\delta_t$
$\hat{A}_t$
$V_\mu(s_t)$
$V_\mu(s_{t+1})$
$L^V(\mu)$
Update $V_\mu$ with the gradient $\nabla L^V(\mu)$ by SGD
Mini-batch Samples (length: $U$)
[<$s_{t_0}, a_{t_0}, r_{t_0+1}, s_{t_0+1}$>,<$s_{t_0+1}, a_{t_0+1}, r_{t_0+2}, s_{t_0+2}$>, …, < $a_{t_0+U-1}, s_{t_0+U-1}, r_{t_0+U}, s_{t_0+U}$ >]

Figure 2.9: A visualization of A2C/PPO. Image credits: (Lim et al., 2020).

This Bellman equation is the starting point for learning an approximator to $Q^*(s,a)$. Suppose the approximator is a neural network $Q_\phi(s,a)$, with parameters $\phi$, and that we have collected a set $\mathcal{D}$ of transitions $(s,a,r,s',d)$ (where d indicates whether state $s'$ is terminal). We can set up a mean-squared Bellman error (MSBE) function, which tells us roughly how closely $Q_\phi$ comes to satisfying the Bellman equation:

$$L(\phi,\mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathrm{E}}\left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d)\max_{a'} Q_\phi(s',a')\right)\right)^2\right] \tag{2.21}$$

To tackle the instability issue, DQN uses experience replay and a target network. In experience replay, the observation sequences $(s_t, a_t, r_t, s_{t+1})$ are stored in a *replay buffer* $\mathcal{D}$, and then experiences are sampled uniformly to train the networks. $\mathcal{D}$ should be large enough to reduce variance, but it may not be always good to contain almost everything. At iteration $i$, the network's parameters are updated based on the gradient w.r.t the network's parameter. The term:

$$r + \gamma(1-d)\max_{a'} Q_\phi(s',a') \tag{2.22}$$

is called the target, because when we minimize the MSBE loss, we are trying to make the Q-function be more like this target.

Figure 2.8 illustrates the learning flow of DQN. A replay memory stores the state transitions. The state $s$ and the action $a$ flow through the Q-network, while the next stage $s'$ flows through the target network. The gradient is updated based on the loss between the Q-value predicted by the Q-network and the Q-value evaluated by the target network, plus the reward $r$ (Equation 2.21).

### 2.2.10 Advantage Actor-Critic

Policy Gradient is a method that directly optimizes the policy. Let $\pi_\theta$ denote a policy with parameters $\theta$, and $J(\pi_\theta)$ denote the expected finite-horizon undiscounted return of the policy. The gradient of $J(\pi_\theta)$ is

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) A^{\pi_\theta}(s_t, a_t)\right] \tag{2.23}$$

where $\tau$ is a trajectory and $A^{\pi_\theta}$ is the advantage function for the current policy.

The policy gradient algorithm works by updating policy parameters via stochastic gradient ascent on policy performance:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_{\theta_k}) \tag{2.24}$$

Policy gradient algorithms compute advantage function estimates based on the infinite-horizon discounted return, despite otherwise using the finite-horizon undiscounted policy gradient formula.

Advantage Actor-Critic uses $A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$, leading to a much lower variance estimate of the policy gradient. Rewrite using Bellman equation:

$$\hat{A}_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{2.25}$$

which requires only one neural network to approximate $V$. With the parameter $\phi$, the value network is updated by minimizing an objective between the V-function and the Reward, typically the MSE:

$$\phi_{k+1} = \arg\min_\phi \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \ell(V_\phi, \hat{R}_\tau) \tag{2.26}$$

Asynchronous Advantage Actor-Critic (A3C) (Mnih et al., 2016) uses multiple workers and updates the global network asynchronously. A2C is an asynchronous variant of A3C and is found to have similar performance.

### 2.2.11 Proximal Policy Optimization

**Background**: Trust Region Policy Optimization (Schulman et al., 2015). Let $\pi_\theta$ denote a policy with parameters $\theta$. The theoretical TRPO update is:

$$\theta_{k+1} = \arg\max_\theta \mathcal{L}(\theta_k, \theta) \text{ s.t. } \bar{D}_{KL}(\theta||\theta_k) \leqslant \delta \tag{2.27}$$

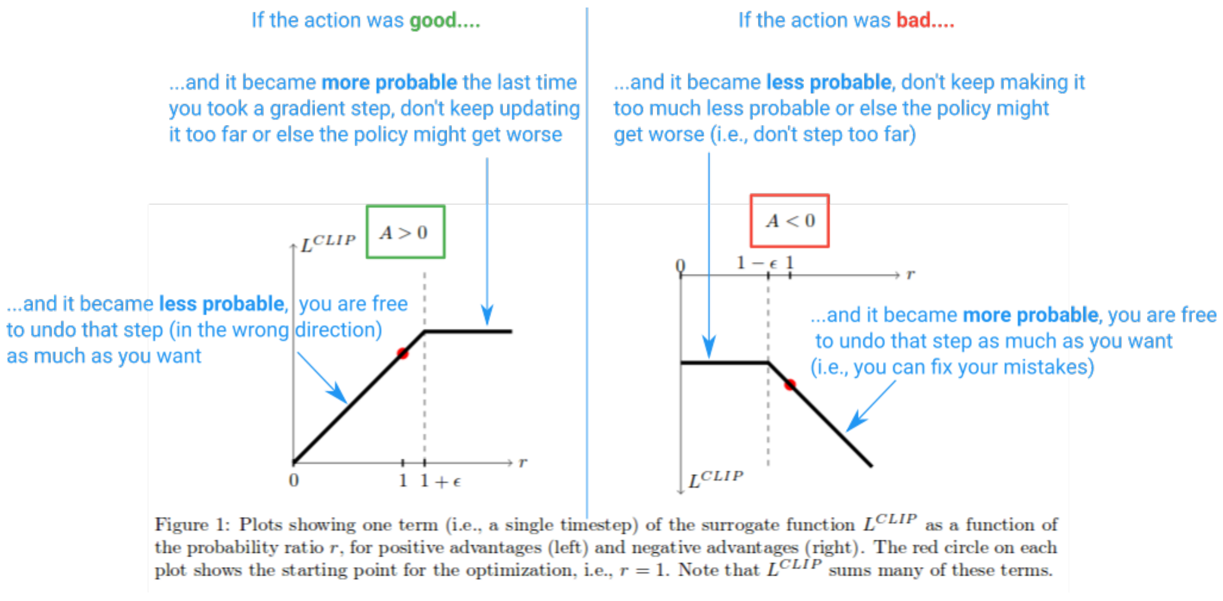
where $\mathcal{L}(\theta_k, \theta)$ is the ”surrogate advantage”, a measure of how policy $\pi_\theta$ performs relative to the old policy $\pi_{\theta_k}$ using data from the old policy:

$$\mathcal{L}(\theta_k, \theta) = \mathop{\mathbb{E}}_{s,a \sim \pi_{\theta_k}} \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a), \tag{2.28}$$

and $\bar{D}_{KL}(\theta||\theta_k)$ is an average KL-divergence between policies across states visited by the old policy:

$$\bar{D}_{KL}(\theta||\theta_k) = \mathop{\mathbb{E}}_{s \sim \pi_{\theta_k}} D_{KL}\left(\pi_\theta(\cdot|s)||\pi_{\theta_k}(\cdot|s)\right) \tag{2.29}$$

Proximal Policy Optimization (Schulman et al., 2017) (PPO) is motivated by the same question as TRPO: how can we make the biggest possible improvement step on a policy using the data we currently have, without stepping so far that we accidentally cause policy collapse, such as catastrophic forgetting.



If the action was **good....**

...and it became **more probable** the last time you took a gradient step, don't keep updating it too far or else the policy might get worse

...and it became **less probable**, you are free to undo that step (in the wrong direction) as much as you want

If the action was **bad....**

...and it became **less probable**, don't keep making it too much less probable or else the policy might get worse (i.e., don't step too far)

...and it became **more probable**, you are free to undo that step as much as you want (i.e., you can fix your mistakes)

Figure 1: Plots showing one term (i.e., a single timestep) of the surrogate function $L^{CLIP}$ as a function of the probability ratio $r$, for positive advantages (left) and negative advantages (right). The red circle on each plot shows the starting point for the optimization, i.e., $r = 1$. Note that $L^{CLIP}$ sums many of these terms.

Figure 2.10: Intuition behind the clip surrogate objective of PPO. Image credits: (Schulman et al., 2017).

TRPO solves this optmization objective using a complicated second-order method, while PPO is a family of first-order methods that use a few other tricks to keep new policies close to old.

PPO-clip updates policies via

$$\theta_{k+1} = \arg\max_{\theta} \underset{s,a\sim\pi_{\theta_k}}{\mathrm{E}} \left[ L(s,a,\theta_k,\theta) \right] \tag{2.30}$$

typically takes multiple steps of gradient descent to maximize the objective. Here $L$ is given by

$$L(s,a,\theta_k,\theta) = \min\left( \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a), \;\; \mathrm{clip}\left( \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon \right) A^{\pi_{\theta_k}}(s,a) \right) \tag{2.31}$$

in which $\epsilon$ is a (small) hyperparameter which roughly says how far away the new policy is allowed to go from the old. Usually $\epsilon = 0.2$. This clip prevents the policy from "collapsing" - that is, the policy turns out to be far worse than expectation.

Figure 2.9 visualizes the learning process of PPO. A trajectory memory stores the interactions of the agent and the environment. After the interaction, a mini-batch is sampled from the memory. The Actor Model calculates the difference between the old and the new policies multiplied by the advantage (Equation 2.28), while the Critic Model calculates the advantage. The Actor Model's parameter is updated via the gradient of the Clip objective (Equation 2.30), while the Critic Model is updated via the gradient of the loss of the Value Function and the Reward (Equation 2.25).

## 2.3 Video Streaming

### 2.3.1 Earrly History

ABR algorithms can be categorized into two main approaches: throughput rate-based and buffer-based. Rate-based algorithms first estimate the network throughput using

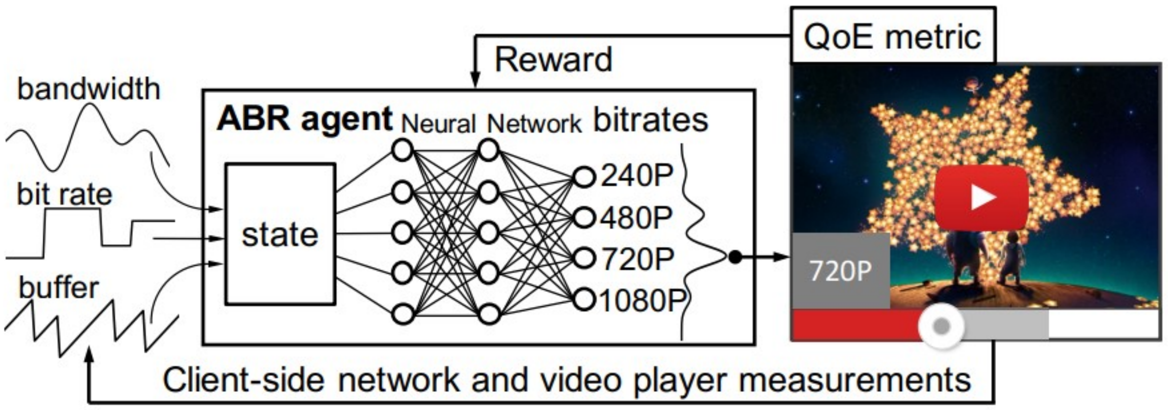


Figure 2.11: Applying RL to ABR by Pensieve. Image credits: (Mao et al., 2017).

variables such as past chunk downloads, past network throughput and then request the quality level as the highest one that the predicted network can handle. For example, (Stockhammer, 2011) predicts the throughput based on the harmonic mean of previous throughput for the past few chunks. Although several efforts aim to improve the performance, throughput-based remains hard in practice (Zou et al., 2015).

Buffer-based approaches consider the buffer occupancy when deciding the quality level of future chunks. The goal is to keep the buffer under a threshold such that the quality level and the rebuffering time are balanced. BOLA (Spiteri et al., 2020) optimizes the I-QoE metric using a Lyapunov objective. Moreover, BOLA also supports abandoning chunk download, whereby a video player can restart a chunk download process at a different quality level if it predicts that rebuffering is unavoidable.

Some works aim to combine these two approaches. MPC (Yin et al., 2015) employs predictive control algorithms that use both throughput estimates and buffer occupancy information to select quality levels that are expected to maximize I-QoE over a horizon of several future chunks. However, MPC heavily depends on the accuracy of the throughput prediction, of which the performance can be degraded significantly (Mao et al., 2017).

A separate line of work is to apply RL to adaptive video streaming. Works by (Claeys et al., 2013, 2014; Chiariotti et al., 2016) apply RL in tabular form, instead of Neural Networks. Tabular RL learns the value function for all possible combinations of states and actions explicitly, which does not generalize well when the state space expands. Pensieve (Mao et al., 2017) applies Deep RL, which uses Neural Networks instead of Tabular learning. Pensieve's algorithms are improved using experiences of the resulting performance of past decisions across a large number of video streaming cycles. This allows Pensieve to improve its policy for various network speed conditions and QoE metrics from data.

### 2.3.2 Problem Statement

A video is divided into several chunks $N$ of equal time segment $\tau$ (e.g.a 240-second video can be divided into $N = 60$ chunks, and each chunk stores $\tau = 4$ seconds of the video). The chunk is further encoded into $L$ quality levels that are distributed among individual streams, which readers might find the levels' names familiar: 720p, 1080p, 1080p@30fps. For the same chunk index, a higher quality level indicates bigger chunk size ($\sigma_{(n,l_1)} < \sigma_{(n,l_2)}$ and $q_{(n,l_1)} < q_{(n,l_2)}$ whenever $l_1 < l_2$). One by one, the client requests a chunk from the server, and the chunk is downloaded to the client. Downloaded chunks are played back to the users. Figure 1.6 best illustrates the encoding and decision process.

| Notation | Definition |
|---|---|
| $q_{(n,l)}$ | Quality score of the chunk index $n$ at quality level $l$. |
| $\sigma_{(n,l)}$ | Size of the chunk index $n$ at quality level $l$. |
| $t$ | Time since start (second). |
| $T$ | Network throughput trace. |
| $T_n$ | Bytes downloaded over the play duration of the n-th chunk. |
| $C_n$ | Average network throughput over the play duration of the n-th chunk. |
| $L$ | Number of quality levels. $L = 7$. |
| $l_n$ | Quality level index. $l \in L$. |
| $n$ | Chunk index. |
| $N$ | Number of chunks in a video. $N = 60$. |
| $x_{(n,l)}$ | Boolean indicator whether the $n$-th chunk at quality level $l$ is selected. |
| $P$ | Number of chunks downloaded before the playback starts. $P = 1$. |
| $\Omega_{min}$ | Minimum buffer threshold (sec). |
| $\Omega_{max}$ | Maximum buffer threshold (sec). $\Omega_{max} = 30$ |
| $\tau$ | Duration of all chunks (sec). $\tau = 4$ |

Table 2.1: Notation and selected values of the variables in this thesis.

Consider a network throughput trace $T(t)$, which is the observed network throughput as a function of time $t$. During any time interval, $(t_1, t_2)$, the area under the curve, $\int_{t_1}^{t_2} T(t)dt$, represents the number of bytes that can be downloaded. For ease of notation, let us denote the number of bytes that can be downloaded over the play duration of the $n$-th chunk given the throughput trace $T$ as $T_n$. For example, $T_2$ is defined as $\int_{t_1}^{(t_1+t_2)} T(t)dt$ (Figure 2.12).

Instead of writing the chunk directly to the storage of the client, the downloaded chunks are stored in the "Replay Buffer" $\Omega$ before being played, which in turn is stored in RAM (Random Access Memory). The replay buffer size is pre-determined depending on the client, but usually it can hold tens of seconds. The replay buffer uses the first-in-first-out rule: chunks that are downloaded first will be played first. The chunk cannot be played until it is fully downloaded to the replay buffer.

The video player requires an initial replay buffer level $\Omega_{min}$ before starting to play, i.e. $P$ chunks are pre-downloaded and will not be used in the optimization objective (usually $P = 1$). When the replay buffer exceeds an upper threshold level $\Omega_{max}$, the video player will stop requesting new chunks. The client must wait for the buffer level to decrease below $\Omega_{max}$ to send the request for a new chunk again. The video player *rebuffers* (freezes) when the chunk index which is going to be played next does not exist in the buffer. The video player then pauses playing and waits for the new chunk until it is completely downloaded.

Adaptive Bitrate (ABR) algorithms aim to optimize the Quality of Experience (QoE) of the users under various circumstances to dynamically choose quality for each video chunk (e.g. 4-second block) based on available observations such as estimated network throughput and the replay buffer. To measure the quality of the ABR algorithm, in this work we focus on two common functions: one proposed by (Yin et al., 2015) and one proposed by (Huang et al., 2019a). In general, both functions have the same two main contributing factors: the quality of the chunk, and the amount of time the video freezes.

(Yin et al., 2015) proposes a QoE metric (which hereby we refer it as **I-QoE** (interruption-allowed QoE)) that maximizes the sum of quality $q$, allows video interruption but penalties

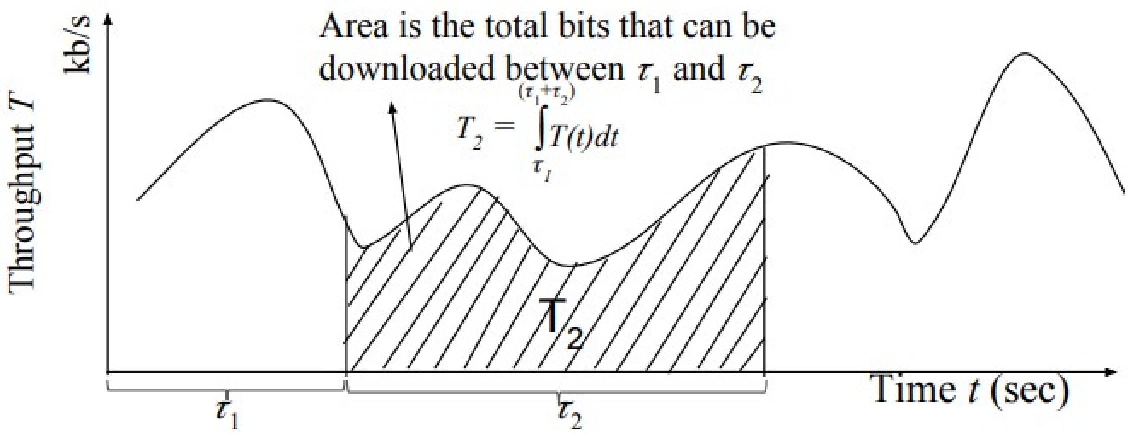


Figure 2.12: A throughput trace. Image credits: (Huang et al., 2019a).

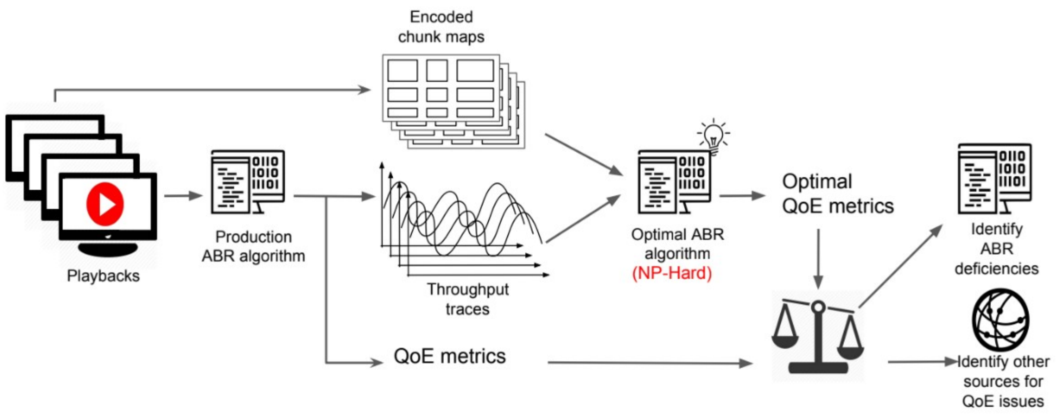


Figure 2.13: Overview of the Video Streaming framework. Finding an optimal QoE metric is NP-Hard, and is expensive to deploy at large scale. Image credits: (Huang et al., 2019a)

the overall QoE if (1) the video is interrupted, (2) if the quality levels of two consecutive chunks switch, and limits the replay buffer size to a fixed threshold. In this work, we focus on this metric, since it is more popular in later works (Mao et al., 2017; Spiteri et al., 2020), can be evaluated at scale (Huang et al., 2019a).

The metric proposed by (Huang et al., 2019a) (which hereby I refer it as **NI-QoE** (non-interrupted QoE)) can be formalized as the maximum sum of quality $\sum q$ that can be downloaded while maintaining an **uninterrupted** video playback (i.e. chunk index $n$ must finish downloading before chunk index $n-1$ finishes playing) but does not limit the replay buffer size (i.e. $\Omega_{max} = \infty$). This problem formulation is known to be a generalization of the NP-Hard Knapsack problem (Kellerer et al., 2004). The generalization will be discussed in Section 2.3.

With the help of the notations in Table 2.1 the problem statement is as follows:

$$
\begin{aligned}
&\text{maximize} \quad \sum_{n=P+1}^{N}\sum_{l=1}^{L} f(x_{(n,l)}q_{(n,l)}) - \lambda \sum_{n=P+1}^{N}\sum_{l=1}^{L} \left|x_{(n,l)}q_{(n,l)} - x_{(n-1,l)}q_{(n-1,l)}\right| \\
&- \mu \sum_{n=P+1}^{N}\sum_{l=1}^{L} \max\left(0, \frac{x_{(n,l)}\sigma_{(n,l)}}{C_n} - \Omega_n\right) \\
&\text{subject to } \Omega_i \leqslant \Omega_{max}, \forall i \\
&\Omega_{i+1} = \max\left(0, \max\left(0, \Omega_i - \frac{\sum_{l=1}^{L} x_{(i,l)}\sigma_{(i,l)}}{C_i}\right) + \tau - \Delta t_i\right), \forall i, \forall l \\
&t_{i+1} = t_i + \frac{\sum_{l=1}^{L} x_{(i,l)}\sigma_{(i,l)}}{C_i} + \Delta t_i, \forall i \\
&T_i = \int_{t_i}^{t_i + t_{i+1} - \Delta t_i} T(t)dt, \forall i \\
&C_i = \frac{T_i}{t_i - t_{i+1} - \Delta t_i}, \forall i \qquad (2.32)\\
&\Delta t_i = \max\left(0, \max\left(0, \Omega_i - \frac{\sum_{l=1}^{L} x_{(i,l)}\sigma_{(i,l)}}{C_i}\right) + \tau - \Omega_{max}\right), \forall i, \forall l \\
&\sum_{l=1}^{L} x_{(n,l)} = 1, \forall n \\
&\Omega_i \in [0, \Omega_{max}], \forall i \\
&x_{(n,l)} \in \{0, 1\}, \forall n, \forall l \\
&\{\lambda, \mu\} \in \mathbb{R} \\
&f = \ln(q_{(n,l)}/q_{(n,0)}) \\
&\text{where the range of } \forall i, \forall n, \forall l \text{ is:} \\
&\forall n \in \{(P+1), ..., N\}, \forall i \in \{(P+1), ..., N\}, \forall l \in \{1, ..., L\}
\end{aligned}
$$

Explanation of Equation 2.32: the objective is to maximize the reward function that consists of three parts: the reward from chunk quality ($\sum_{n=P+1}^{N}\sum_{l=1}^{L} f(x_{(n,l)}q_{(n,l)})$), penalty due to quality switch ($\lambda \sum_{n=P+1}^{N}\sum_{l=1}^{L} \left|x_{(n,l)}q_{(n,l)} - x_{(n-1,l)}q_{(n-1,l)}\right|$) and penalty due to rebuffering ($\mu \sum_{n=P+1}^{N}\sum_{l=1}^{L} \max\left(0, \frac{x_{(n,l)}\sigma_{(n,l)}}{C_n} - \Omega_n\right)$) such that the replay buffer must not exceed a threshold every chunk ($\Omega_i \leqslant \Omega_{max}$). $T_i$ denotes the bytes download over the duration of the $i$-th chunk, and $C_i$ is the average network speed during that duration (thus $\sum_{l=1}^{L} x_{(i,l)}\sigma_{(i,l)}/C_i$ is the download time in seconds). The replay buffer size is determined as the previous size of itself, minus the download time of the chunk and add the chunk's duration. For a more detailed explanation and intuition behind the metric, readers can refer to (Yin et al., 2015).

### 2.3.3 NI-QoE as a generalization of Knapsack

While I-QoE allows video rebuffering, (Huang et al., 2019a) defines the video streaming problem as an interrupted experience as follows:

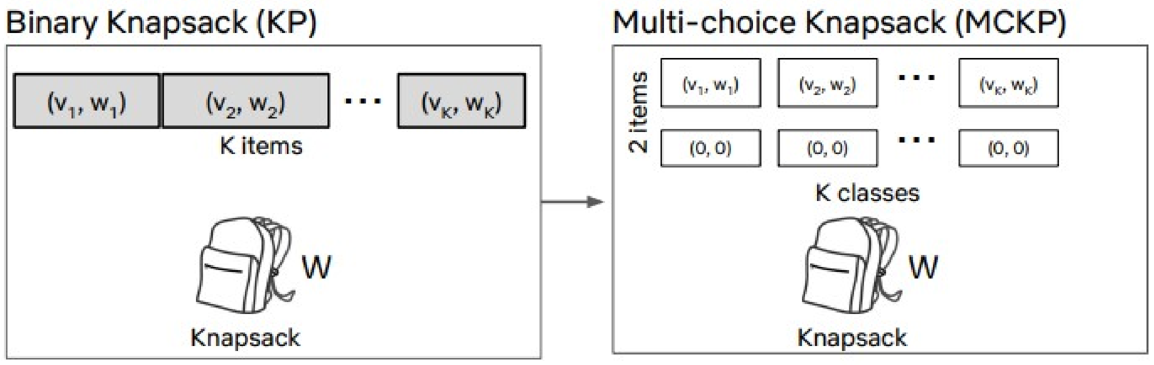


Figure 2.14: MCKP as a generalization of KP. Image credits: (Huang et al., 2019a).

$$\begin{aligned}
&\text{maximize} \sum_{n=P+1}^{N} \sum_{l=1}^{L} x_{(n,l)} q_{(n,l)} \\
&\text{subject to} \sum_{n=P+1}^{i} \sum_{l=1}^{L} x_{(n,l)} \sigma_{(n,l)} \leqslant \sum_{j=1}^{(i-1)} T_j, \forall i \in \{(P+1), ..., N\} \\
&\sum_{l=1}^{L} x_{(n,l)} = 1, \forall n \in \{(P+1), ..., N\} \\
&x_{(n,l)} \in \{0,1\}, \forall n \in \{(P+1), ..., N\}, \forall l \in \{1, ..., L\}
\end{aligned} \tag{2.33}$$

Explanation: the objective is to maximize the sum of the quality of all chunks, such that for every chunk index $n$, it must be downloaded before the chunk index $n-1$ finishes playing (thus no playback interruption is allowed), so the total size of chunks $P+1$ to $n$ must be smaller than the cumulative available bandwidth before chunk index $n-1$ (line 2 in Equation 2.33).

The work also demonstrates that finding the optimal NI-QoE metric is NP-Hard because it is a generalization of the *Multi-Choice Multi-Period Knapsack* (MMKP), a special case of the *Multiple-Choice Nested Knapsack* (MCNKP) problem mentioned by (Miller et al., 2013). MMKP is simply a generalization of the Multi-Choice Knapsack problem (MCKP), which in turn generalizes the original knapsack problem (KP). The original KP problem is known to be NP-Hard (Kellerer et al., 2004). In the following section, I summarize the study in (Huang et al., 2019a).

**MMKP is NP-Hard:** First, consider the traditional binary Knapsack problem. Consider a knapsack of capacity $W$ and a set of $K$ items, each with weight $w_i$ and value $v_i$, the objective is to choose a set of items such that they maximize the total value but the total weight must not exceed $W$:

$$\begin{aligned}
&\text{maximize} \sum_{i=1}^{K} v_i x_i \\
&\text{subject to} \sum_{i=1}^{K} w_i x_i \leqslant W, x_i \in \{0,1\}
\end{aligned} \tag{2.34}$$

The Multi-Choice Knapsack problem (MCKP) is a generalization of the KP problem, where the set of items is partitioned into classes (like video chunks are encoded into multiple quality levels) (Kellerer et al., 2004). Consider $M$ mutually disjointed classes

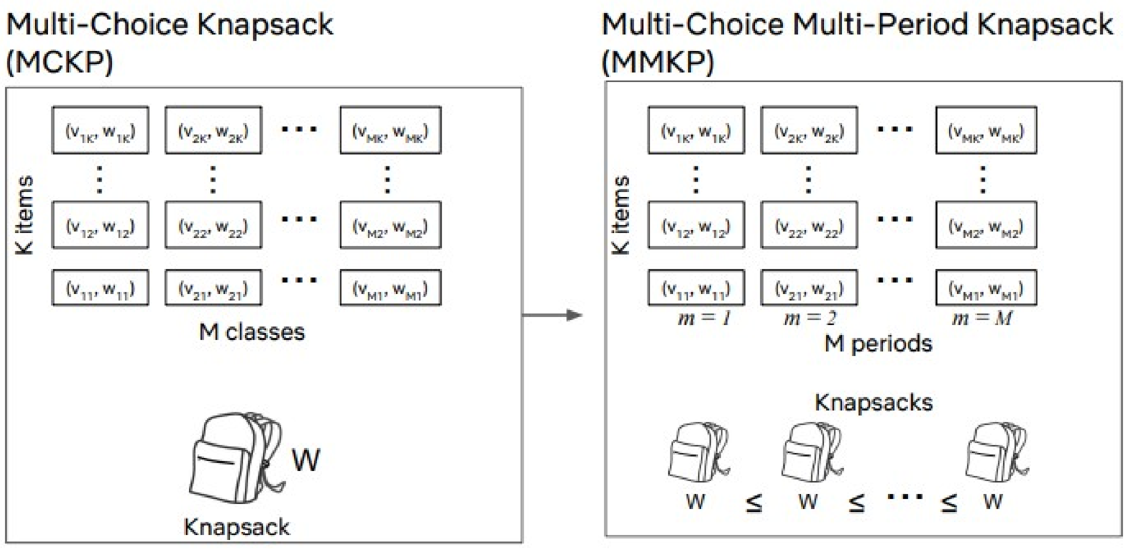


Figure 2.15: MCKP as a generalization of MMKP. Image credits: (Huang et al., 2019a).

$N_1, ..., N_M$ of items to be packed into a knapsack of capacity $W$. Each item $j \in N_i$ has a value $v_{ij}$ and weight $w_{ij}$. The objective is to choose **one** item from each class such that the sum of the items' values does not exceed $W$. Let $x_{ij} \in \{0, 1\}$ denotes whether item $j$ is chosen in class $N_i$. MCKP is NP-Hard because KP is a reduction of MCKP. The problem is formulated as follows:

$$\begin{aligned}
&\text{maximize } \sum_{i=1}^{M} \sum_{j \in N_i} v_{ij} x_{ij} \\
&\text{subject to } \sum_{i=1}^{M} \sum_{j \in N_i} w_{ij} x_{ij} \leqslant W, \\
&\sum_{j \in N_i} x_{ij} = 1, i \in \{1, ..., M\} \\
&x_{ij} \in \{0, 1\}, i \in \{1, ..., M\}, j \in N_i
\end{aligned} \tag{2.35}$$

Multiple-Choice Multi-Period Knapsack (MMKP) (Lin and Wu, 2004) is a generalization of the MCKP problem. MMKP has items scattered in $M$ periods, and its knapsack capacity expands over the periods to accommodate additional items. The objective is to select one item from each period and to maximize the overall value covering these periods. The problem formulation is as follows:

$$\begin{aligned}
&\text{maximize } \sum_{i=1}^{M} \sum_{j \in N_i} v_{ij} x_{ij} \\
&\text{subject to } \sum_{k=1}^{i} \sum_{j \in N_t} w_{kj} x_{kj} < W_i, \forall i \in 1, ..., M \\
&\sum_{j \in N_i} x_{ij} = 1, i \in 1, ..., M \\
&x_{ij} \in \{0, 1\}, i \in \{1, ..., M\}, j \in N_i \\
&W_1 \leqslant W_2 \leqslant ... \leqslant W_M
\end{aligned} \tag{2.36}$$

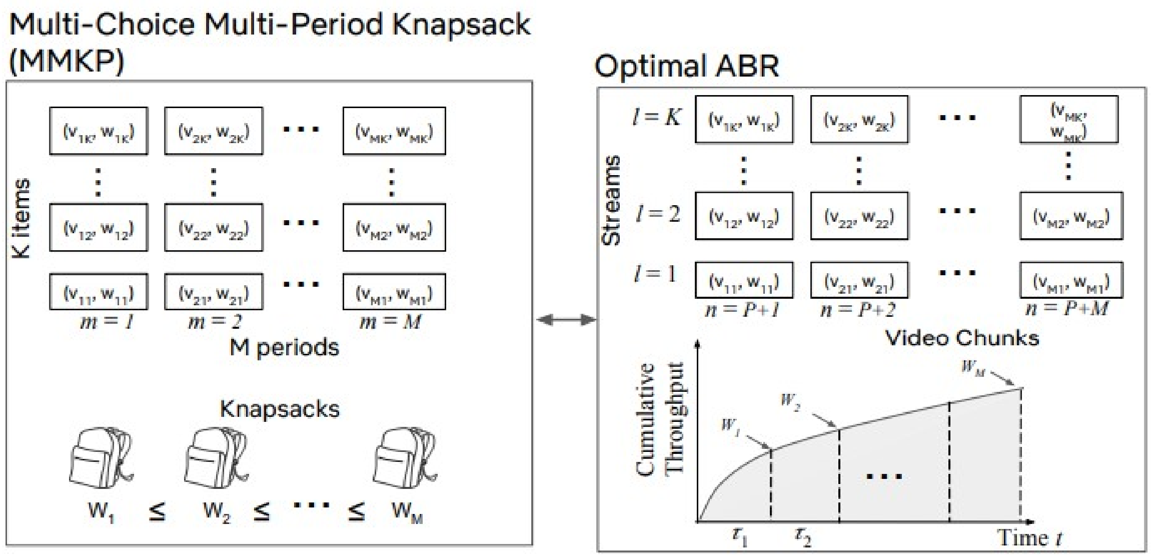


Figure 2.16: MMKP as a generalization of the optimal ABR. Image credits: (Huang et al., 2019a).

We now can convert the optimal ABR algorithm as a MMKP problem by letting $T_i = W_i, T_i = W_i - W_{i-1}, 1 < i < N$, $N = P + M$ and $L = K$. Figure 2.16 illustrates the generalization, which then shows that they are equivalent problems.

## 2.3.4 I-QoE as a similar metric

It can be seen that the optimization objective of the problem introduced in Equation 2.32 is somewhat similar to Equation 2.33. However, at the current state, I make no assumption nor formulation about the difficulty and the type of the problem, whether it is NP-Hard or a generalization of MMKP, or it is a type of Mixed Integer-(Non)Linear Problem. (Huang et al., 2019a) suggests that adding the replay buffer constraint to equation 2.33 transforms it to the Mixed Integer Linear Problem.

# Chapter 3

# Methodology

## 3.1 PyTorch

PyTorch[1] (Paszke et al., 2019) is an open-source machine learning framework that accelerates the path from research prototyping to production deployment. PyTorch provides two high-level features: (1) Tensor computing (like NumPy) with strong acceleration via GPU, and (2) Deep neural networks built on a type-based automatic differentiation system. PyTorch is trending in the research community due to its flexibility, and most RL libraries built on PyTorch allow full customization.

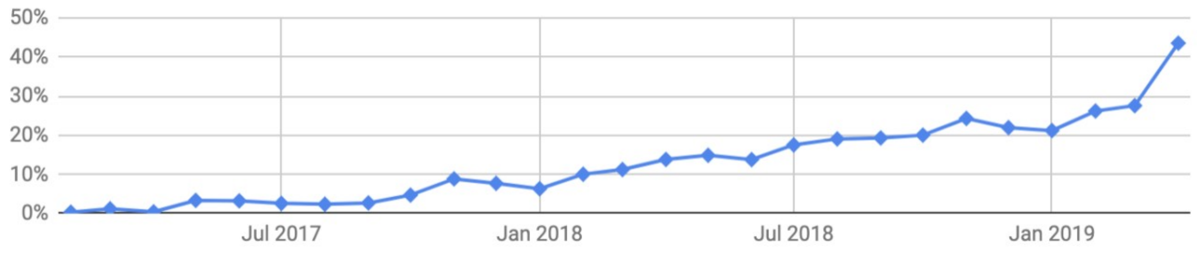


Figure 3.1: ArXiv papers that mention PyTorch over the years. Image credits: (Paszke et al., 2019).

Compared to other frameworks, PyTorch has the following advantages:

- Dynamic vs Static: While PyTorch uses dynamic computation graphs, TensorFlow uses static computation graphs. With the release of TensorFlow 2.0 there has been a major shift towards eager execution, and away from static graph computation. However, this major shift has caused incompatibility in various codebases, making TensorFlow an unstable choice if any similar incident occurs. PyTorch is a better choice.

- Data Parallelism: PyTorch uses asynchronous execution of Python to implement data parallelism. With TensorFlow you need to manually configure every operation for data parallelism.

- Various research libraries are built on PyTorch (e.g., Stable-Baselines3 (Raffin et al., 2021)). It is not the same case with TensorFlow.

[1]https://pytorch.org/

## 3.2 OpenAI Gym Environment

Gym (Brockman et al., 2016) is an interface for developing RL environments. It supports from classical controls to Atari games. As an environment interface, it is compatible with any RL library.

An environment interface consists of two main functions: `step` and `reset`. At the beginning of each episode, `reset` is called, which resets all variables within an episode. After that, `step` is called consecutively until the episode terminates, and the process repeats. The `step` function requires input as the action of that step, and returns the next state/observation, a scalar value as a reward, and a boolean variable indicating whether the episode is terminated.

The design of OpenAI Gym is based on the OpenAI's researchers experiences in developing and comparing RL algorithms, and the experience using other benchmark suites. The design decision of Gym can be summarized as follows:

- **Environments, not agents**. Two core factors in RL are the agent and the environment. Gym has chosen to only provide an interface for the environment to maximize convenience for users to implement custom environments of their choices.
- **Emphasize computational cost, not just performance**. The performance of an RL algorithm can be calculated based on two factors: the final performance and the amount of time it takes to learn, as known as the computational cost. Since additional amount of computation can be used to improve the final performance, Gym also takes care of measuring this aspect.
- **Encourage ideas and reviews, not performance competition.** Although the OpenAI Gym website establishes a leaderboard to compare algorithms, the aim of it is not to create a competition, but rather to share custom environments and research directions, and to be a meaningful benchmark for different RL algorithms.
- **Strict versioning for environments**. Small changes in the environment can lead to drastically different results. To avoid this, Gym guarantees that any changes to an environment will be denoted by a version number instead of a direct replacement.

## 3.3 Stable-Baseline3

Stable-Baselines3 (SB3) is an open-source framework implementing seven commonly used model-free deep RL algorithms (Raffin et al., 2021). The library takes great care employ design patterns best practices to achieve high-quality implementations that are easy to extend while matching prior results before being published. Each algorithm is evaluated on common environments and compared to prior implementations. The code test covers 90%+ of the code and ensures that any implementation errors are minimized. In March 2022, SB3 had 3.1k+ stars on GitHub, 500+ closed issues and 200+ pull requests, making it one of the most popular RL libraries.

The features of Stable-Baselines 3 can be summarized as follows:

- **Simple API**. Training agents in Stable-Baselines3 takes just a few lines of code, allowing users to easily use the implemented algorithms and components in their experiments, as well as apply RL to various tasks and interesting environments.

- **Documentation**. SB3 comes with extensive documentation of the code API. It also includes a user guide, covering both basic and more advanced users with a collection of concrete examples.
- **High-Quality Implementations**. Algorithms are evaluated against published results by comparing the agent learning curves. Continuous integration checks that all changes pass unit tests and type checks, as well as validating the code style and documentation.
- **Comprehensive**. Stable-Baselines3 contains popular state-of-the-art on- and off-policy algorithms. Moreover, SB3 provides various algorithm-independent features. The authors support logging to files and TensorBoard. To speed up training, it also supports parallel (or "vectorized") environments based on the original baselines.

Using a library instead of implementing from scratch has several advantages:

1. It saves time and effort.
2. The algorithms in the library are intensively benchmarked to match the published result in various tasks, thus they are reliable.
3. The simulation environment is required to follow the Gym interface (Brockman et al., 2016), making it easier to be used by other algorithms and easier to extend, even if they come from a different library.
4. Works by (Engstrom et al., 2020; Henderson et al., 2018; Islam et al., 2017; Hu et al., 2021; Irpan, 2018; Andrychowicz et al., 2021) pointed out that small implementation details (e.g. normalization, global gradient clipping) can greatly influence the overall performance. In fact, removing these details makes these algorithms' performance almost the same (Engstrom et al., 2020; Hu et al., 2021). These details are widely considered to be "implementation tricks", which cause bias in comparison.

## 3.4 Implementation Details

In this section, I describe the implementation details of the simulated environment. Then, I describe the datasets, libraries and tuning strategy that are used in this thesis.

### 3.4.1 States, Actions, and Rewards

**States**: define one step as the action to choose the quality of the next chunk. At each step $i$, the agent observes the following variables:

- A vector of previous network speed $\{C_{i-1}, ..., C_{i-6}\}$ in Mbps.
- A vector of the chunk's size $\{\sigma_{(i,0)}, ..., \sigma_{(i,L)}\}$ in Mbps.
- A vector of previous delay time $\{\Delta t_{i-1}, ..., \Delta t_{i-6}\}$ in seconds.
- A scalar replay buffer size before downloading the chunk $\Omega_{i-1}$, normalized by 10.
- A numeric scalar value indicates the percentage of played chunks $i/N \in [0, 1]$, increasing each step.

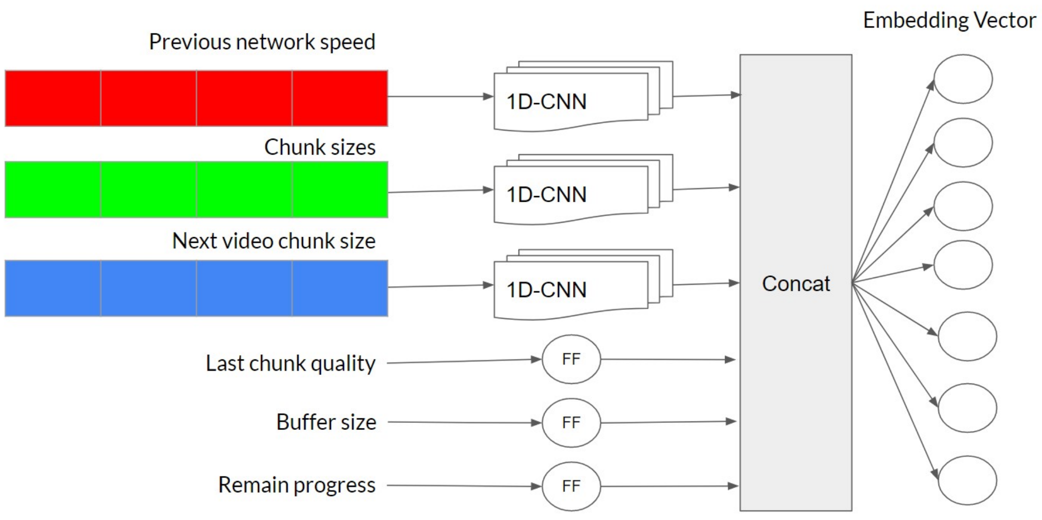


Figure 3.2: Simplified architecture of the Features Extractor Network.

- A categorical scalar value last chunk quality level $q_{(i-1,\cdot)}$, which is encoded to a one-hot vector.

I use a simple network to extract the inputs into an embedding vector like (Mao et al., 2017), which is described in Figure 3.2. Namely:

- The scalars are passed through feedforward layers of 128 units with ReLU activation function.
- The arrays are passed through 1D-CNN layers of 128 filters with kernel size 4 and stride 1, followed by ReLU.
- Except for the last chunk quality: it is encoded to a one-hot vector, and then is passed through a feedforward layer of 128 units, followed by ReLU.

These output values are flattened and concatenated to form an embedding vector, then they are passed through two layers with Tanh activation function in between. Finally, the Features Extractor produces an embedding vector, which is passed to the policy and/or the value networks. For faster computing time, the Features Extractor is shared between the policy and/or value networks. Figure 3.3 visualizes the flow of the training process.

**Action**: the action is to pick a scalar quality level $l$ from $L$ available levels for the client to request that chunk level to the server.

**Reward**: The environment returns a scalar value "Reward" as the I-QoE metric (objective in Equation 2.32), normalized by 100.

**Trajectory**: a trajectory's length is 60 steps, which is equal to the number of chunks defined earlier ($N = 60$).

### 3.4.2 Datasets

**Video and Quality levels**: For the video, the Elephant Dream video dataset is used (Blender, 2014). It is encoded into 20 different quality levels of 4 seconds for each chunk. The following 7 encoding bitrate: $\{700, 900, 2000, 3000, 5000, 6000, 8000\}$ Kbps are chosen, following the guidelines by (Google, 2021), which translates to approximately the following human-friendly quality levels: (240p, 360p, 480p, 720p, 720p@60fps, 1080p,

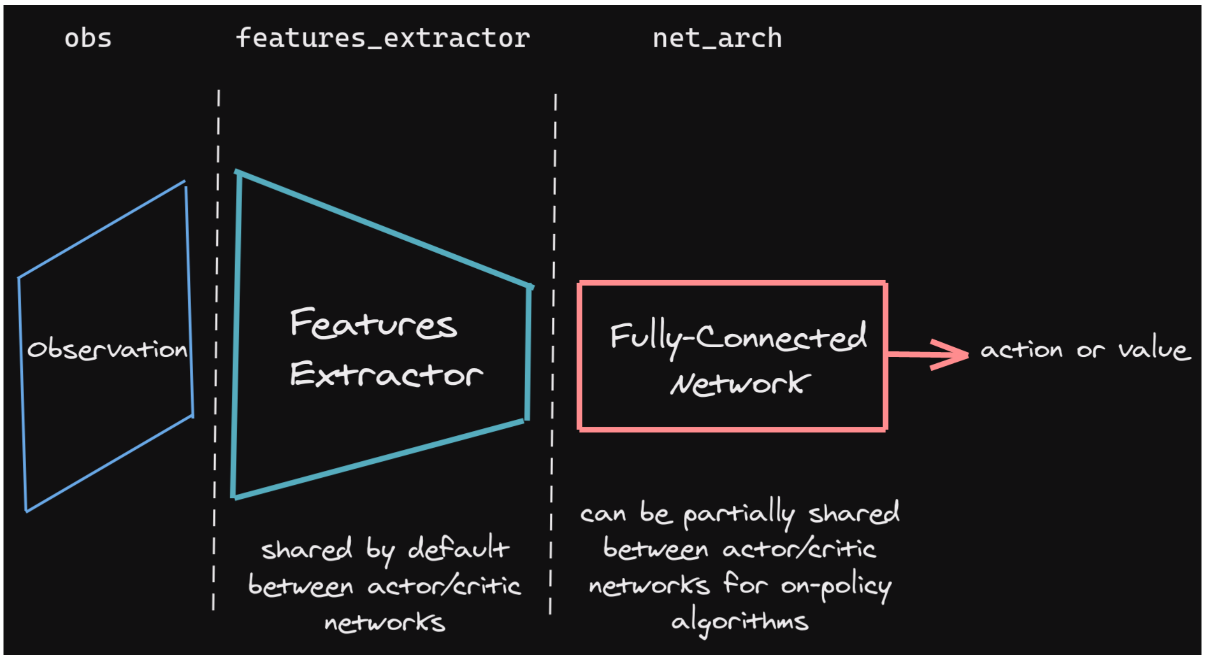

obs
features_extractor
net_arch
Observation
Features Extractor
Fully-Connected Network
action or value
shared by default between actor/critic networks
can be partially shared between actor/critic networks for on-policy algorithms

Figure 3.3: Training Flow. Inputs are passed through the Features Extractor. The embedding vector is then passed through the Actor/Critic/Q-Network. Image credits: (Raffin et al., 2021).

1080p@60fps). Thus, the agent has 7 discrete actions for each step. The first 60 chunks of the video ($N = 60$) are used, which is 240 seconds. The default quality at the beginning for the pre-downloaded $P$ chunks is the lowest quality.

**4G LTE**: the 4G LTE dataset (Raca et al., 2018) contains 135 traces, with an average duration of 15 minutes per trace, at 1-second granularity. This dataset collected traces from Irish mobile operators, with 5 mobility patterns (static, pedestrian, car, bus, and train). By using 60 seconds sliding window across this dataset, I also generated 1,000 traces with 320 seconds per trace.

**FCC**: The FCC dataset contains over 1 million throughput traces, at a granularity of 10 samples per second (FCC, 2019). I construct 1,000 random traces (each spanning 320 seconds) for our dataset. Although the method of collecting data is very similar to Pensieve, our throughput traces are randomly selected from the "download speed" category instead of using "web browsing" as in Pensieve. According to (FCC, 2019), the throughput in "web browsing" is measured by taking the sum of HTML content size and all resources size divided by fetch time, which is the time consumed to download that HTML and these resources. However, because the size of web pages varies, the fetch time cannot ensure to be fixed. In the case of the "download speed" category, the measurement operation time is fixed in 10 seconds. The client attempts to download as much as possible and the average throughput achieved during this period is recorded. In this way, the network segment will be ensuring granularity in 10 seconds. I use the September 2019 collection.

### 3.4.3 Miscellaneous

**Training and testing sets.** In both datasets, I randomly split 80% of the dataset to train, and 20% of the dataset to test. At the beginning of every episode, the trace and the pointer are picked randomly from the train set. The pointer is incremented every step. If the pointer reaches the end, the trace resets itself back to the first index. For more robust training, I concatenate the FCC and the LTE datasets to train the RL agents. Each agent is trained for $885,000$ steps, which is $15,000$ episodes.

**Random seed.** To ensure a fair and rigorous evaluation, I follow evaluation guidelines suggested by (Colas et al., 2019) and (Henderson et al., 2018) as follows: after training the agent, I evaluate the agent on an unseen test set (which is split like described above) for 200 episodes and record the mean of reward. The trial is repeated 10 times and uses the Welch's t-test implemented in Scipy (Virtanen et al., 2020) and reports the p-value returned. If the p-value is less than a threshold, it can be said that it is statistically significant. The validation set's input order is fixed during evaluation, *i.e.* each algorithm observes the same input at each step.

**Library.** As noted by (Engstrom et al., 2020; Henderson et al., 2018; Islam et al., 2017; Hu et al., 2021; Irpan, 2018; Andrychowicz et al., 2021), code-level optimization tricks (e.g., observation normalization, reward scaling, global gradient clipping) from different codebases heavily impact the performance of DRL algorithms. Thus, I use the implemented algorithms in Stable-Baselines3 (Raffin et al., 2021), without modifying *any* part of the algorithm. PPO and A2C can be trained with 4 parallel environments, while DQN is not supported. For PPO and DQN, Adam optimizer is used (Kingma and Ba, 2015). For A2C, I also add the option to use RMSprop (Tieleman and Hinton, 2012) or Adam as a hyperparameter.

**Hyperparamter tunning.**. Each algorithm is run for $\approx 150$ trials with a different seed, each from a wide range of hyperparameter distributions. After that, I use the best hyperparameter combination found so far to run the algorithms 10 more times and report the average of the results. More information about the hyperparameter tuning strategy can be found in Appendix A.

**$\mu$ and $\lambda$**: Following (Mao et al., 2017), I set $\lambda = 1$ and test two cases regarding $\mu$: $\mu \in \{2.66, 5.0\}$.

# Chapter 4

# Result and Evaluation

## 4.1 Other algorithms

I compare the DRL-based rate adaptation methods with some other non-DRL methods, *i.e.*, random, constant, throughput-based (Stockhammer, 2011) and BOLA (Spiteri et al., 2020). Particularly,

- Random (RAN): at each step, the next quality is picked randomly.
- Constant (CON): at each step, the next quality is picked as the 3000Kbps quality (which is known as HD (720p) quality).
- Throughput-based (THRB): at each step, the download chunk's size has the highest quality level which is smaller than that the harmonic mean of previously downloaded 3 chunks' sizes.
- BOLA: is the buffer-based adaptation method that used Lyapunov objective to minimize rebuffering and also maximize video quality. I re-implement the BOLA-U version.

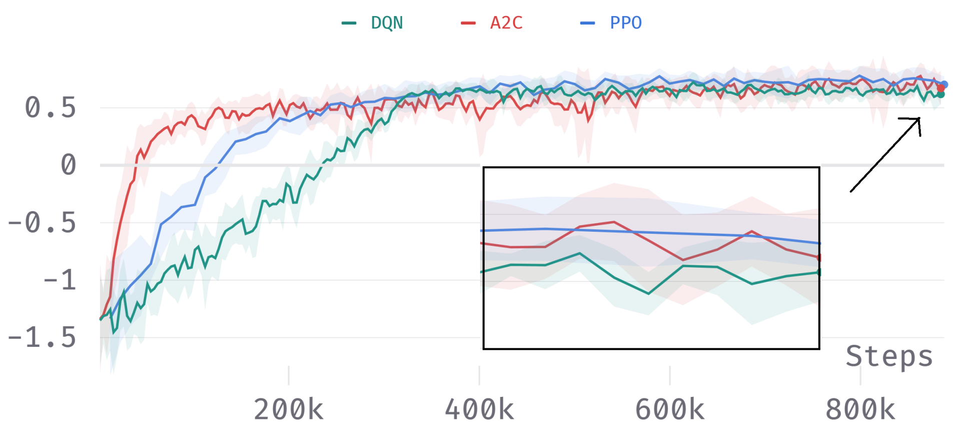


Figure 4.1: Training curve with $\mu = 2.66$.

## 4.2 Training curve

I first present the results with the best hyperparameters in Table 4.1 with case $\mu = 2.66$. Figures 4.1 show the convergence of three algorithms with about 900,000 training steps. I take the average of results as the running average of 100 steps (shaded areas). The algorithms converge around the 400k-th step.

I present the detailed result in Table 4.1. From the table, PPO outperforms A2C and DQN on the FCC dataset, but on the LTE dataset, there is no difference between them. However, if we take the rebuffering time into account, DQN has the shortest rebuffering time.

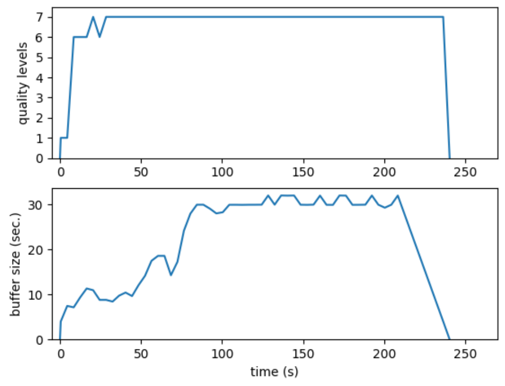
Table 4.1: I-QoE metrics on the test set of both datasets in case $\mu = 2.66$. Higher is better. Each algorithm is averaged across 10 seeds. Bold indicates the best result.

| FCC | I-QoE | Utility | Quality-switch | Rebuffering | Rebuf time (s) |
|---|---|---|---|---|---|
| A2C | $0.903 \pm 0.104$ | 1.044 | 0.094 | 0.046 | 1.729 |
| DQN | $0.898 \pm 0.047$ | 1.014 | 0.094 | 0.020 | 0.751 |
| PPO | $\mathbf{0.971} \pm 0.046$ | 1.080 | 0.077 | 0.031 | 1.165 |
| THRB | $0.752 \pm 0.439$ | 0.933 | 0.178 | 0.002 | 0.075 |
| BOLA | $0.843 \pm 0.505$ | 0.957 | 0.062 | 0.051 | 1.917 |
| RAN | $-2.296 \pm 0.069$ | 0.785 | 0.581 | 2.500 | 50.00 |
| CON | $-4.224 \pm 0.001$ | 1.160 | 0.020 | 5.364 | 107.23 |
| **LTE** | **I-QoE** | **Utility** | **Quality-switch** | **Rebuffering** | **Rebuf time (s)** |
| A2C | $0.600 \pm 0.083$ | 0.966 | 0.106 | 0.260 | 9.774 |
| DQN | $\mathbf{0.616} \pm 0.046$ | 0.915 | 0.145 | 0.152 | 5.741 |
| PPO | $0.590 \pm 0.044$ | 0.978 | 0.136 | 0.251 | 9.436 |
| THRB | $0.537 \pm 0.672$ | 0.841 | 0.194 | 0.267 | 10.03 |
| BOLA | $0.513 \pm 0.942$ | 1.029 | 0.145 | 0.371 | 13.94 |
| RAN | $-1.304 \pm 0.023$ | 0.786 | 0.582 | 1.508 | 56.70 |
| CON | $-1.713 \pm 0.001$ | 1.160 | 0.020 | 2.854 | 107.28 |

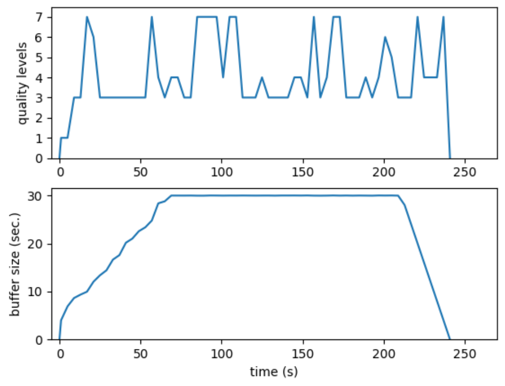
I further test the three algorithms in the case $\mu = 5$[1]. Intuitively, (1) it is reasonable to sacrifice some quality for shorter rebuffering time, and (2) good algorithms should be robust to this change. I present the results in Table 4.2. PPO shows a significantly higher I-QoE metric in this case, while the rebuffering time is maintained the lowest, and is reduced compared to the case $\mu = 2.66$.

Figure 4.2 shows the quality level of PPO, BOLA, and THRB on a randomly chosen FCC trace. It can be seen that PPO serves the best quality by utilizing buffer size effectively. On the other hand, BOLA chooses low quality and THRB serves unstable quality.

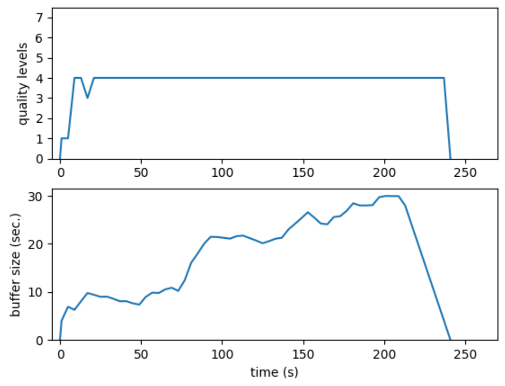
[1]The curve of this case is similar to Figure 4.1, so it is not shown here.



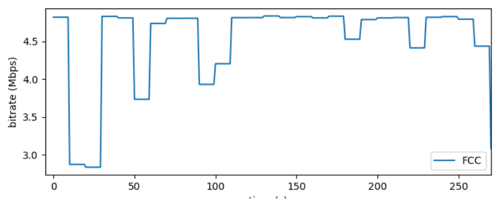
(a) PPO



(b) THRB



(c) BOLA



(d) bandwidth

Figure 4.2: Chosen quality levels and buffer size over time on randomly picked network traces from the FCC dataset. Quality level 0 means ”not playing”.

Table 4.2: I-QoE metrics on the test set of both datasets in case $\mu = 5$. Each algorithm is averaged across 10 seeds.

| FCC | I-QoE | Utility | Quality-switch | Rebuffering | Rebuf time (s) |
|---|---|---|---|---|---|
| A2C | $0.576 \pm 0.383$ | 0.819 | 0.071 | 0.172 | 3.44 |
| DQN | $0.854 \pm 0.068$ | 1.018 | 0.118 | 0.044 | 0.88 |
| PPO | $\mathbf{0.957} \pm 0.053$ | 1.057 | 0.069 | 0.031 | 0.62 |
| THRB | $0.750 \pm 0.445$ | 0.933 | 0.178 | 0.005 | 0.10 |
| BOLA | $0.798 \pm 0.626$ | 0.957 | 0.062 | 0.096 | 1.92 |
| RAN | $-2.296 \pm 0.030$ | 0.786 | 0.582 | 2.500 | 50.00 |
| CON | $-4.091 \pm 0.002$ | 1.160 | 0.019 | 5.232 | 104.63 |
| LTE | I-QoE | Utility | Quality-switch | Rebuffering | Rebuf time (s) |
| A2C | $0.242 \pm 0.304$ | 0.745 | 0.085 | 0.418 | 8.36 |
| DQN | $0.419 \pm 0.095$ | 0.928 | 0.162 | 0.346 | 6.92 |
| PPO | $\mathbf{0.462} \pm 0.048$ | 0.948 | 0.127 | 0.357 | 7.14 |
| THRB | $0.444 \pm 1.064$ | 0.841 | 0.194 | 0.202 | 4.04 |
| BOLA | $0.189 \pm 1.525$ | 1.029 | 0.145 | 0.694 | 13.88 |
| RAN | $-2.657 \pm 0.049$ | 0.786 | 0.582 | 2.835 | 56.70 |
| CON | $-4.224 \pm 0.002$ | 1.160 | 0.020 | 5.364 | 107.28 |

Table 4.3: Welch's t-test on the experiments. Bold indicates a p-value less than 0.05.

| | FCC($\mu = 2.66$) | | FCC($\mu = 5$) | | LTE($\mu = 2.66$) | | LTE($\mu = 5$) | |
|---|---|---|---|---|---|---|---|---|
| Order | t-stat | p-value | t-stat | p-value | t-stat | p-value | t-stat | p-value |
| A2C vs DQN | 0.130 | 0.897 | -2.260 | **0.048** | -0.539 | 0.598 | -1.750 | 0.108 |
| PPO vs A2C | 3.464 | **0.002** | 3.107 | **0.012** | -0.330 | 0.746 | 2.261 | **0.048** |
| PPO vs DQN | 1.883 | 0.083 | 3.714 | **0.001** | -1.292 | 0.212 | 1.295 | 0.217 |

## 4.3 Welch's t-test

The differences between the algorithms are small, making it hard to determine whether they are significant. Therefore, I conduct Welch's t-test, which is calculated using 10 runs with different random seeds, where each run uses the mean of the reward on 200 test episodes as the sample. By default, the null hypothesis assumes two populations have identical mean values (i.e., they are sampled from the same distribution), and the alternative hypothesis assumes two populations have different mean values.

From Table 4.3, we observe that only for the case of the LTE dataset with $\mu = 2.66$, the tests have near-zero t-statistics values with a large p-value. On the other hand, for other cases, PPO significantly outperforms the other two with very high t-statistics and low p-value. If we take four other algorithms into consideration, PPO is still the best algorithm with the lowest or second-lowest rebuffering time.

## 4.4 Runtime Analysis

PPO and A2C use two neural networks, while DQN uses only the Q-network. Moreover, their update rules are different from each other, thus it is important to compare the algorithms' runtime to determine the practical value. From the empirical experiments, each algorithm takes approximately 2.5 hours on a 4 cores Intel XEON CPU @2.3GHz and without GPU. Each algorithm is tuned for 120 trails. In total, the runtime is approximately 38 CPU days. The runtime difference between CPU and GPU is not significant.

Overall, from Figures 4.3, 4.4, 4.5 we can see that PPO is stable across a wide range of hyperparameters. Thus, I suggest to use PPO as a baseline algorithm.

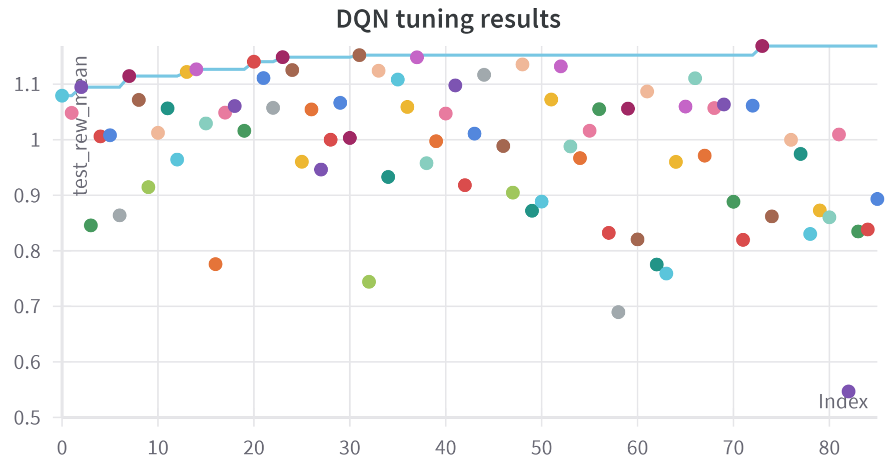


Figure 4.3: Scatter plot of DQN tuning results.

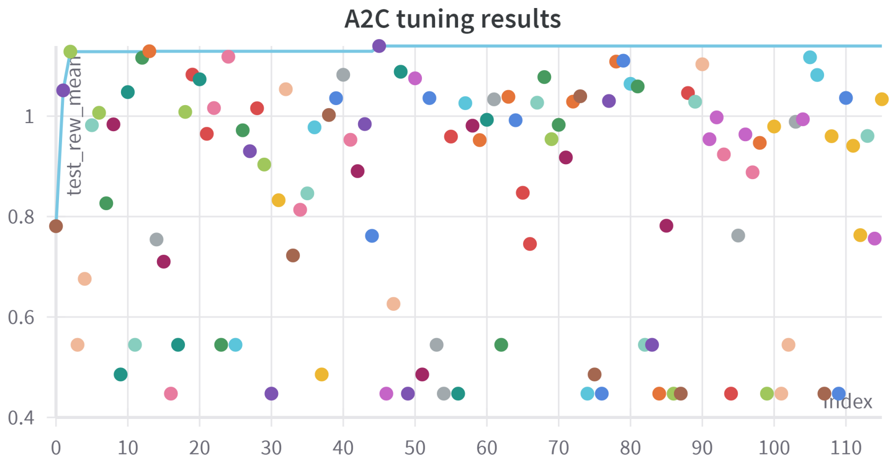


Figure 4.4: Scatter plot of A2C tuning results.

## 4.5 Random Search

I use Grid Search to search for the I-QoE for the case $\mu = 2.66$. For each network throughput trace, it is searched for 3000 times. The results show that, for FCC, I-QoE is 0.40 and for LTE, I-QoE is 0.31. Although this procedure takes about 7 hours to complete

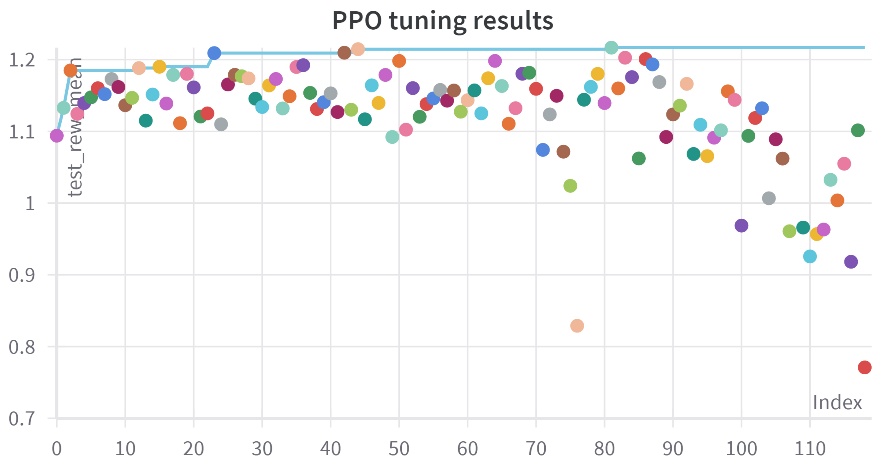


Figure 4.5: Scatter plot of PPO tuning results.

(on the same CPU spec) and is impractical to use in the real world, it serves as a baseline to compare other algorithms.

## 4.6 Open-source code

Aside from the code that is attached to this thesis's submission, I also open-source the training and tuning monitors in the links below for future readers to use as references:

Final training: https://wandb.ai/ntnghia_iu/singlepath_newenv
Tuning: https://wandb.ai/aeryss/singlepath-tuning

# Chapter 5

# Conclusion and Future Work

## 5.1 Conclusion

In this thesis, my goal is to present to the readers the essence of applying Reinforcement Learning-based Adaptive Bitrate Algorithms. First, I introduce the history and development of related background and formulates the problem of this thesis. Then, I introduce the problem formulation, and summarize the findings in (Huang et al., 2019a), in which the authors described that two problem are similar and can be NP-Hard. After that, I describe the methodology: the libraries and frameworks that are used in this thesis, as well as the implementation details. Following the best evaluation practices such as Welch's t-test based on multiple trials, I present the results, which show that that RL-based ABR algorithms have the prospects to outperform traditional algorithms.

## 5.2 Future Work

In this section, I propose some future goals for this direction of research, in decreasing order of importance according to myself:

- Reproducing the Dynamic Programming algorithm in (Spiteri et al., 2020; Mao et al., 2017) to determine the optimal QoE of *any* algorithm. This is the most important goal, since unlike accuracy which lies between 0 and 1, the optimal QoE has no upper nor lower constraints.

- A unified benchmark suite for ABR. I could not find any good benchmark suite for ABR at the time of writing. While such works exist (Mao et al., 2019; Huang, 2019; Huang et al., 2019b), the API it provides does not follow the Gym interface, thus making it not possible to use implemented RL algorithms.

- Determine the problem's family formulated in 2.32. Determining the problem's family will unlock the use of other areas' algorithms to improve ABR algorithms.

- Real-time simulation. Due to time constraints, I could not implement the real-time simulation integrated in DASH in time. It would be best to deploy the algorithm in a real demo, in which all the decisions are visualized in real-time.

- Extend the idea to Federated Learning to enhance Data Privacy. It has been shown that Federated Learning can converge in the supervised learning manner (Li et al., 2020), but to the best of my knowledge, it is not well known nor researched in the Reinforcement Learning area. A good starting point is (Li et al., 2021b).

# Bibliography

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

Nilesh Barla. Understanding representation learning with autoencoder: Everything you need to know about representation and feature learning, 2021.

Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021.

Xianfeng Tang, Boqing Gong, Yanwei Yu, Huaxiu Yao, Yandong Li, Haiyong Xie, and Xiaoyu Wang. Joint modeling of dense and incomplete trajectories for citywide traffic volume inference. *The World Wide Web Conference on - WWW '19*, 2019. doi: 10.1145/3308558.3313621. URL http://dx.doi.org/10.1145/3308558.3313621.

Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 197–210, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450346535. doi: 10.1145/3098822.3098843. URL https://doi.org/10.1145/3098822.3098843.

Chaitanya Ekanadham. Using machine learning to improve streaming quality at netflix, 2018. URL https://netflixtechblog.com/using-machine-learning-to-improve-streaming-quality-at-netflix-9651263ef09f. Accessed: 7th December 2021.

Fei-Fei Li, Ranjay Krishna, and Danfei Xu. CS231n: Convolutional Neural Networks for Visual Recognition, 2021a. URL https://github.com/cs231n/cs231n.github.io. Accessed: 7th December 2021.

Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition*. O'Reilly Media, Inc., 2019. ISBN 9781492032649.

Haohan Wang, Bhiksha Raj, and Eric P. Xing. On the origin of deep learning. *CoRR*, abs/1702.07800, 2017. URL http://arxiv.org/abs/1702.07800.

Joshua Achiam. Spinning up in deep reinforcement learning, 2018. URL https://spinningup.openai.com/en/latest/.

Xinyuan Zhou, Peng Wu, Haifeng Zhang, Weihong Guo, and Yuanchang Liu. Learn to navigate: Cooperative path planning for unmanned surface vehicles using deep reinforcement learning. *IEEE Access*, PP:1–1, 11 2019. doi: 10.1109/ACCESS.2019.2953326.

Hyun-Kyo Lim, Ju-Bong Kim, Joo-Seong Heo, and Youn-Hee Han. Federated reinforcement learning for training control policies on multiple iot devices. *Sensors*, 20:1359, 03 2020. doi: 10.3390/s20051359.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

Te-Yuan Huang, Chaitanya Ekanadham, Andrew J. Berglund, and Zhi Li. Hindsight: Evaluate video bitrate adaptation at scale. In *Proceedings of the 10th ACM Multimedia Systems Conference*, MMSys '19, page 86–97, New York, NY, USA, 2019a. Association for Computing Machinery. ISBN 9781450362979. doi: 10.1145/3304109.3306219. URL https://doi.org/10.1145/3304109.3306219.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html. GitHub: https://github.com/DLR-RM/stable-baselines3.

Dan Cireşan, Ueli Meier, Jonathan Masci, and Jürgen Schmidhuber. Multi-column deep neural network for traffic sign classification. *Neural Networks*, 32:333–338, 2012. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2012.02.023. URL https://www.sciencedirect.com/science/article/pii/S0893608012000524. Selected Papers from IJCNN 2011.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 1097–1105, Red Hook, NY, USA, 2012. Curran Associates Inc.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation, 2015.

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

Li Xin. Statistics of acceptance rate for the main ai conferences, 2021. URL https://github.com/lixin4ever/Conference-Acceptance-Rate. Accessed: 7th December 2021.

Klaus Schwab. *The Fourth Industrial Revolution*. Crown Publishing Group, USA, 2017. ISBN 1524758868.

Arthur .E. Bryson. Optimal control-1950 to 1985. *IEEE Control Systems Magazine*, 16 (3):26–33, 1996. doi: 10.1109/37.506395.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, Jun 2013. ISSN 1076-9757. doi: 10.1613/jair.3912. URL http://dx.doi.org/10.1613/jair.3912.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, 20–22 Jun 2016. PMLR. URL https://proceedings.mlr.press/v48/mniha16.html.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/schulman15.html.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109.

Ekin Dogus Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation policies from data. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. URL https://arxiv.org/pdf/1805.09501.pdf.

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2017.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2017.

Long M. Luu, Zeyi Huang, and Haohan Wang. Automixup: Learning mix-up policies with reinforcement learning. In *International Conference on Machine Learning (ICML) ML4data Workshop*, 2021. URL https://github.com/minhlong94/minhlong94/blob/main/AutoMixup.pdf.

Long M. Luu, Zeyi Huang, Haohan Wang, Dong Huang, Yong Jae Lee, and Eric P. Xing. Expeditious saliency-based mixup through random gradient thresholding, 2022. URL https://bit.ly/3tLqedy.

Matteo Gadaleta, Federico Chiariotti, Michele Rossi, and Andrea Zanella. D-dash: A deep q-learning framework for dash video streaming. *IEEE Transactions on Cognitive Communications and Networking*, 3(4):703–718, 2017. doi: 10.1109/TCCN.2017.2755007.

Samuel Reason. Music streaming actually existed back in 1890, 2020. URL https://blitzlift.com/music-streaming-actually-existed-back-in-1890/. Accessed:

7th December 2021.

GrandViewResearch. Video streaming market size, share & trends analysis report by streaming type, by solution, by platform, by service, by revenue model, by deployment type, by user, by region, and segment forecasts, 2021 - 2028, 2020. URL https://www.grandviewresearch.com/industry-analysis/video-streaming-market. Accessed: 7th December 2021.

Christopher Harper. Facebook gaming vs. twitch vs. youtube gaming: What's the best live game streaming platform?, 2021. URL https://www.maketecheasier.com/best-live-game-streaming-platform/. Accessed: 7th December 2021.

Sandvine. Global internet phenomena report: North america and latin america, 2018.

S. Shunmuga Krishnan and Ramesh K. Sitaraman. Video stream quality impacts viewer behavior: Inferring causality using quasi-experimental designs. In *Proceedings of the 2012 Internet Measurement Conference*, IMC '12, page 211–224, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450317054. doi: 10.1145/2398776.2398799. URL https://doi.org/10.1145/2398776.2398799.

Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. Understanding the impact of video quality on user engagement. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, page 362–373, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450307970. doi: 10.1145/2018436.2018478. URL https://doi.org/10.1145/2018436.2018478.

Te-Yuan Huang, Nikhil Handigol, Brandon Heller, Nick McKeown, and Ramesh Johari. Confused, timid, and unstable: Picking a video streaming rate is hard. In *Proceedings of the 2012 Internet Measurement Conference*, IMC '12, page 225–238, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450317054. doi: 10.1145/2398776.2398800. URL https://doi.org/10.1145/2398776.2398800.

Xuan Kelvin Zou, Jeffrey Erman, Vijay Gopalakrishnan, Emir Halepovic, Rittwik Jana, Xin Jin, Jennifer Rexford, and Rakesh K. Sinha. Can accurate predictions improve video streaming in cellular networks? In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, HotMobile '15, page 57–62, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450333917. doi: 10.1145/2699343.2699359. URL https://doi.org/10.1145/2699343.2699359.

Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over http. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, volume 45-4, page 325–338, New York, NY, USA, August 2015. Association for Computing Machinery. doi: 10.1145/2829988.2787486. URL https://doi.org/10.1145/2829988.2787486.

Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack Problems*. Springer, Berlin, Germany, 2004.

Thomas Stockhammer. Dynamic adaptive streaming over http– standards and design principles. In *Proceedings of the second annual ACM conference on Multimedia systems*, pages 133–144, 2011.

Kevin Spiteri, Rahul Urgaonkar, and Ramesh K. Sitaraman. Bola: Near-optimal bitrate adaptation for online videos. *IEEE/ACM Transactions on Networking*, 28(4):1698–1711, 2020. doi: 10.1109/TNET.2020.2996964.

Praveen Kumar Yadav, Arash Shafiei, and Wei Tsang Ooi. Quetra: A queuing theory approach to dash rate adaptation. In *Proceedings of the 25th ACM International Conference on Multimedia*, MM '17, page 1130–1138, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349062. doi: 10.1145/3123266.3123390. URL https://doi.org/10.1145/3123266.3123390.

Akamai. dash.js. https://github.com/Dash-Industry-Forum/dash.js/, 2016.

Vladimir Vapnik. Principles of risk minimization for learning theory. In J. Moody, S. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4. Morgan-Kaufmann, 1992. URL https://proceedings.neurips.cc/paper/1991/file/ff4d5fbbafdf976cfdc032e3bde78de5-Paper.pdf.

Vitaly Feldman, Venkatesan Guruswami, Prasad Raghavendra, and Yi Wu. Agnostic learning of monomials by halfspaces is hard. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 385–394, 2009. doi: 10.1109/FOCS.2009.26.

David E. Rumelhart, Geoffrey Hinton, and James L. McClelland. *Learning Internal Representations by Error Propagation*, volume ICS Report 8506. 1987.

George V. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314, 1989.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(89)90020-8. URL https://www.sciencedirect.com/science/article/pii/0893608089900208.

Geogre Cybenko. Continuous valued neural networks with two hidden layers are sufficient. 1988.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249.

Anssi Kanervisto, Christian Scheller, and Ville Hautamäki. Action space shaping in deep reinforcement learning. *IEEE Conference on Games*, 2020.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

Maxim Claeys, Steven Latré, Jeroen Famaey, Tingyao Wu, Werner Van Leekwijck, and Filip De Turck. Design of a Q-learning-based client quality selection algorithm for HTTP adaptive video streaming. In *Adaptive and Learning Agents Workshop, part of AAMAS2013, Proceedings*, pages 30–37, 2013.

Maxim Claeys, Steven Latré, Jeroen Famaey, Tingyao Wu, Werner Van Leekwijck, and Filip De Turck. Design and optimisation of a (fa)q-learning-based http adaptive streaming client. *Connection Science*, 26(1):25–43, 2014. doi: 10.1080/09540091.2014.885273. URL https://doi.org/10.1080/09540091.2014.885273.

Federico Chiariotti, Stefano D'Aronco, Laura Toni, and Pascal Frossard. Online learning adaptation strategy for dash clients. In *Proceedings of the 7th International Conference on Multimedia Systems*, MMSys '16, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342971. doi: 10.1145/2910017.2910603. URL https://doi.org/10.1145/2910017.2910603.

Konstantin Miller, Nicola Corda, Savvas Argyropoulos, Alexander Raake, and Adam Wolisz. Optimal adaptation trajectories for block-request adaptive video streaming. In *2013 20th International Packet Video Workshop*, pages 1–8, 2013. doi: 10.1109/PV.2013.6691452.

Edward Lin and Chung-Min Wu. The multiple-choice multi-period knapsack problem. *Journal of The Operational Research Society - J OPER RES SOC*, 55:187–197, 02 2004. doi: 10.1057/palgrave.jors.2601661.

Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep rl: A case study on ppo and trpo. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=r1etN1rtPB.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. 2018. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16669.

Riashat Islam, Peter Henderson, Maziar Gomrokchi, and Doina Precup. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. In *Reproducibility in Machine Learning Workshop (ICML)*, 2017. URL https://arxiv.org/pdf/1708.04133.pdf.

Jian Hu, Siyang Jiang, Seth Austin Harding, Haibin Wu, and Shih wei Liao. Rethinking the implementation tricks and monotonicity constraint in cooperative multi-agent reinforcement learning, 2021.

Alex Irpan. Deep reinforcement learning doesn't work yet. https://www.alexirpan.com/2018/02/14/rl-hard.html, 2018.

Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphaël Marinier, Leonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. What matters for on-policy deep actor-critic methods? a large-scale study. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=nIAxjsniDzg.

Blender. Elephants dream movie, 2014. URL https://orange.blender.org/.

Google. Choose live encoder settings, bitrates, and resolutions  youtube help, 2021. URL https://support.google.com/youtube/answer/2853702.

Darijo Raca, Jason J. Quinlan, Ahmed H. Zahran, and Cormac J. Sreenan. Beyond throughput: A 4g lte dataset with channel and context metrics. In *Proceedings of the 9th ACM Multimedia Systems Conference*, MMSys '18, page 460–465, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450351928. doi: 10.1145/3204949.3208123. URL https://doi.org/10.1145/3204949.3208123.

FCC. The tenth measuring broadband america fixed broadband report: A report on consumer fixed broadband performance in the united states, 2019.

Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. A hitchhiker's guide to statistical comparisons of reinforcement learning algorithms, 2019.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Tijmen Tieleman and Geoffey Hinton. Lecture 6.5—RMSProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

Hongzi Mao, Parimarjan Negi, Akshay Narayan, Hanrui Wang, Jiacheng Yang, Haonan Wang, Ryan Marcus, ravichandra addanki, Mehrdad Khani Shirkoohi, Songtao He, Vikram Nathan, Frank Cangialosi, Shaileshh Venkatakrishnan, Wei-Hung Weng, Song Han, Tim Kraska, and Mohammad Alizadeh. Park: An open platform for learning-augmented computer systems. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/f69e505b08403ad2298b9f262659929a-Paper.pdf.

Tianchi Huang. Abr-dqn. https://github.com/godka/ABR-DQN, 2019.

Tianchi Huang, Shunkun Huang, Kasim Te, and Zhang Rui-Xiao. Pensieve-ppo. https://github.com/godka/Pensieve-PPO, 2019b.

Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of fedavg on non-iid data. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HJxNAnVtDS.

Qinbin Li, Zeyi Wen, Zhaomin Wu, Sixu Hu, Naibo Wang, Yuan Li, Xu Liu, and Bingsheng He. A survey on federated learning systems: Vision, hype and reality for data privacy and protection. *IEEE Transactions on Knowledge and Data Engineering*, page 1–1, 2021b. ISSN 2326-3865. doi: 10.1109/tkde.2021.3124599. URL http://dx.doi.org/10.1109/TKDE.2021.3124599.

Lukas Biewald. Experiment tracking with weights and biases, 2020. URL https://www.wandb.com/. Software available from wandb.com.

# Appendix A

# Hyperparameter Tuning Strategy

In this appendix, I describe the hyperparameter tuning strategy. More details about the hyperparameters can be found in each paper (Mnih et al., 2013, 2016; Schulman et al., 2017) [1].

I use Weights & Biases (Biewald, 2020) for experiment tracking and visualizations to develop insights for this thesis. Weights & Biases offers the Bayesian tuning strategy to search for the optimal hyperparameters. For each algorithm, I perform 120 trials with different seeds each and use the hyperparameter from the best trial to train the final model 10 times for each algorithm. Hyperparameters are sampled from a uniform distribution. Due to the implementation, PPO and A2C can run 4 parallel environments, while DQN is not supported yet. These choices of hyperparameters are picked arbitrarily.

Table A.1: Hyperparameters tuning range of DQN.

| **Hyperparameter** | **Type** | **Sampling Distribution** |
|---|---|---|
| Learning rate | float | U(1e-5,1e-3) |
| Experience replay buffer size | int | U(59, 11800) |
| Batch size | int | U(59, 590) |
| Learning starts | int | U(295, 2360) |
| Discount factor | float | {0.95, 0.99} |
| Polyak coef | float | {0.99, 1.0} |
| Train frequency | int | U(30, 120) |
| Gradient steps | int | U(-1, 59) |
| Target update interval | int | U(30, 200) |
| Exploration fraction | float | U(0.2, 0.6) |

In addition to the algorithm-specific hyperparameters above, each algorithm also has the following network hyperparameters, which is presented in Table A.4 [2].

I present the final hyperparameters in this experiment in Tables A.5 and A.6.

[1]For PPO, trajectory size = Batch size × N steps coef

[2]The features dim also determines the two last layers of the Features Extractor network: $(dim \times 2 \longrightarrow dim)$ with Tanh in between

Table A.2: Hyperparameters tuning range of A2C.

| **Hyperparameter** | **Type** | **Sampling Distribution** |
|---|---|---|
| Learning rate | float | U(1e-5,1e-3) |
| N steps coef | int | U(5, 590) |
| Epoch | int | {10, 20, 30} |
| Discount factor | float | {0.99, 1.0} |
| GAE coef | float | {0.95, 0.99, 1.0} |
| Entropy coef | float | {0, 1e-5, 1e-8} |
| Value function coef | float | U(0.2, 0.5) |
| Use RMSprop | bool | true, false |
| Normalize advantages | bool | true, false |

Table A.3: Hyperparameters tuning range of PPO.

| **Hyperparameter** | **Type** | **Sampling Distribution** |
|---|---|---|
| Learning rate | float | U(1e-5,1e-3) |
| Batch size | int | U(59, 590) |
| N steps coef | int | U(1, 5) |
| Epoch | int | {10, 20, 30} |
| Discount factor | float | {0.99, 1.0} |
| GAE coef | float | {0.95, 0.99} |
| Clip range | float | {0.2, 0.3} |
| Entropy coef | float | {0, 1e-5, 1e-8} |
| Value function coef | float | U(0.2, 0.5) |

Table A.4: Shared hyperparameters range across all algorithms.

| **Hyperparameter** | **Type** | **Sampling Distribution** |
|---|---|---|
| Features dim | int | {128, 256, 512} |
| Policy network units | int | {64, 128, 256, 512} |
| Policy network layers | int | U(1, 4) |
| Value network units | int | {64, 128, 256, 512} |
| Value network layers | int | U(1, 4) |
| Activation function | str | {tanh, relu} |

Table A.5: Final hyperparamters of DQN

| Hyperparameter | Value |
|---|---|
| Learning rate | 0.00042 |
| Experience replay buffer size | 11340 |
| Learning starts | 1655 |
| Batch size | 181 |
| Polyak update coef | 0.99 |
| Discount factor | 0.99 |
| Model update step (train_freq) | 96 |
| Gradient step | 12 |
| Target Network update interval | 102 |
| Exploration Fraction | 0.36757 |
| Initial exploration rate | 1.0 |
| Final exploration rate | 0.05 |
| Gradient clipping value | 10 |
| Optimizer | Adam |
| Features dim | 128 |
| Activation function | ReLU |
| Q-network units | 64 |
| Q-network layers | 1 |

Table A.6: Hyperparameters of A2C (left) and PPO (right)

| Hyperparameter | Value |
|---|---|
| Learning rate | 0.00011 |
| Experiences per update | 7 |
| Discount factor | 1 |
| GAE lambda | 0.95 |
| Entropy coef | 1e-8 |
| Value function coef | 0.32803 |
| Gradient clipping value | 0.5 |
| Use RMSProp | True |
| Normalize advantage | True |
| Features dim | 256 |
| Activation function | Tanh |
| Policy network units | 64 |
| Policy network layers | 1 |
| Value network units | 128 |
| Value network layers | 1 |

| Hyperparameter | Value |
|---|---|
| Learning rate | 0.00012 |
| Experiences per update | 2875 |
| Batch size | 556 |
| N steps coef | 5 |
| Update epoch | 10 |
| Discount factor | 1 |
| GAE lambda | 0.95 |
| Clip range | 0.3 |
| Entropy coef | 1e-8 |
| Value function coef | 0.43487 |
| Gradient clipping value | 0.5 |
| Optimizer | Adam |
| Features dim | 512 |
| Activation function | ReLU |
| Policy network units | 512 |
| Policy network layers | 4 |
| Value network units | 128 |
| Value network layers | 3 |