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

APPLICATION OF REINFORCEMENT LEARNING IN DYNAMIC VIDEO STREAMING

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

Ho Chi Minh City, Vietnam
2020

APPLICATION OF REINFORCEMENT LEARNING IN DYNAMIC ADAPTIVE STREAMING OVER HTTP

APPROVED BY:

________________________________ ,
Võ Thị Lưu Phương, Ph.D

____
Nguyễn Thị Thanh Sang, Ph.D

THESIS COMMITTEE
(Whichever applies)

ACKNOWLEGMENTS

I would like to express deep gratitude and appreciation toward Dr. Võ Thị Lưu Phương and Dr. Nguyễn Thị Thanh Sang for their wonderful encouragement and support throughout the entirety of this study

I also want to thanks the faculty of the School of Computer Science of the International University for their support and guidance along my four years in the university

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

ABR:	Bitrate Adaptive Algorithm
API:	Application Programing Interface
BOLA:	Buffer Occupancy based Lyapunov Algorithm
DASH:	Dynamic Adaptive Streaming Over HTTP
DASH-MPEG:	Dynamic Adaptive Streaming Over HTTP
QoE:	Quality of Experience
RL:	Reinforcement Learning
SVM:	Support vector machine

ABSTRACT

Video streaming is the most popular service all around the world. On an annual report in 2019, Sandvine, a leading provider of network intelligence solutions, stated that more than 60% of traffic on the internet is originated from video streaming services (Sandvine, 2019). This figure, however, is expected to increase even more in the upcoming years due to the transition from HD video bitrate to UHD or 4K video streaming as the number of 4K TV connection increase according to Cisco report (Cisco, 2020). This explosion in popularity is an opportunity but also raises many questions for service providers. One of which is the challenge of maintain high quality of experience (QoE) for customer across a massive range of device, network condition and bitrate quality. This is especially challenging due to customer desired to have high video bitrate while capable of avoiding lag and re-buffering which are, in general, conflict goals. In this thesis I would like to have a look at a popular solution for the prior problem as well as discussing about the various methods that being used or develop that going with the solution

CHAPTER 1
INTRODUCTON

1.1. Problem statement

A widely adopted technique that helps on resolving this issue is DASH-MPEG. DASH-MPEG have to mechanism that especially useful against the aforementioned challenge. On one hand, video is broken up into many small segments with various bitrate that can be download separately which offer a wider range of options on video qualities and avoiding wasting bandwidth and loading time for customer whom only watch the video partially. Furthermore, DASH-MPEG utilized a buffer that is available on clients' devices in order to deal with unexpected situations, mainly sudden changed, drops in network bandwidth while having a baseline to decide on should higher bitrate being selected and delivered to customer. The part which makes the decision on which segments being downloaded and also the third and final component of DASH is the bitrate adaption algorithm (ABR). This also will be the main focus of this thesis

1.2. Existing solution

Currently, the two most used ABR in DASH-MPEG are Throughput Rule and Bola Rule. Both ABR have an implementation on Dash.js (Spiteri, Sitaraman, & Sparacio, 2018) as well as an extra implementation combining both ABRs strength which is called Dynamic. These rule using a combination of network features to try to make an accurate prediction on which segment should be chosen in order to maximize QoE.

For Throughput Rule, the primary feature in concern is the overall network throughput. Using an implementation of a sliding window Throughput Rule try to predict upcoming throughput (usually the average of the prior moments multiply by a save threshold) and choose a segment that the predicted throughput can facilitate.

Bola, on the other hand, determines the next video segment according to the buffer level currently available. The higher bitrate level will not be chosen until there are enough stored segments in buffer to support it. Bola does not take into account the network value

All of the previously mentioned solution is currently being used in the real world to dealing with the issue of adaptive streaming. Due to the long research history combine with human controlled parameter, these methods are, in some way, reliable and robust. However, that does not mean that there is no way to improve them. Recent years have seen a surge in artificial intelligent study and application in real life thanks to the scientific advances that make computer faster than ever. In the field of video streaming, there is also no exception.

One study as such is HASBRAIN (Sieber, Hagn, Kellerer, Moldovan, & Hoßfeld, 2018). This study focus on the use of a traditional technique in the field of artificial intelligent called SVM in order to categorized different network situation, and then use an simple neuron network to learn from the categorized network and match the network situation to a predetermined set of reaction obtains using dynamic programming. The neuron network is later used as an ABR.

There also studies on more recently popular AI technique. The Q-learning client (Claeys, Latr´e, Famaey, & Turck, 2014) and the PENSIEVE (Mao, Netravali, & Alizadeh, 2017) both utilize reinforcement learning in their ABR agent. The prior paper used a tabular version of q-learning in order to train the agent. This is possible thanks to a discretization data into multiple states and let the agent to interact and pick the most suitable action corresponding to each state. The later, also the direct inspiration for this thesis, uses actor-critic algorithm to learn the environment instead. By applying this approach, there is no need to actually discretize the data and the agent have full control on how to asset and evaluate data obtained from the environment. A neuron network is used to capture this knowledge and later used as an ABR

1.3. Scope and objective

In this thesis, we will explore the use of reinforcement learning in MPEG-DASH. A various technique and test will be performed and asset to learn the impact on final result. Construction of neuron network and learning techniques will be loosely based on PENSIEVE but with the adoption of the new library Tensorflow as well as some slight modification to learning mechanic. The focus on this thesis is only to evaluate the learning process's effectiveness and all data and graph in this thesis only based on a network simulation instead of applying to real network situation. Although that maybe an interesting work of future to apply to the actual network and video player

1.4. Thesis structure

There are six chapters in this thesis: introduction, literature review, methodology, implementation and result, discussion and evaluation, conclusion and future work

Chapter 1: Introduce the problem concerned by this thesis and how it is intended to be solved

Chapter 2: Provide details mechanism of MPEG-DASH, its solution, how reinforcement learning works as well as related work in AI that used in this thesis

Chapter 3: Describe the experiment set up, data

Chapter 4: Training process as well as the final resulted ABR

Chapter 5: Make comparison and discuss on the meaning of the received results

Chapter 6: Conclude on the result of the thesis and make suggestion on future works that can be done

CHAPTER 2

LITERATURE REVIEW

2.1. Adaption method in ABR

The Throughput rule:

As mentioned prior, throughput rule rely on recent history throughput make a prediction of the network condition in the upcoming download session. The prediction then will be scaled down by a safe threshold (typically 90%) to produce the safe threshold which will be used to pick the next suitable video segment, one that the network can download within the time the previous segment being played or at least produce the least buffered or lagged. If the network condition is excellent and the video player can stock up a large amount of buffer, this ABR will continue to download high quality video segment despite the drop in network as long as the buffer level is above a certain “rich buffer level”.

This set up allow the video player to react quickly to the increase throughput in the network environment. As a result video segment usually has higher quality when compare to other method but suffer majorly from lag, re-buffer and high quality switch frequency which are especially popular in situation where throughput fluctuate a lot or unstable connection like in live event or on mobile 3G/4G and wifi network

The BOLA rule (Buffer Occupancy based Lyapunov Algorithm)

Unlike throughput rule, BOLA rule (Spiteri, Urgaonkar, & Sitarama, BOLA: Near-Optimal Bitrate Adaptation for Online Videos, 2016) utilizes the video player buffer as the most important channel of input for making decision on the next suitable video segment. The mechanism and mathematical reasoning underlying the algorithm is a little bit more complicate therefore quite some explanation is needed

Suppose the video is divided into consecutive, non-overlapping slots (segments) that are indexed k={1,23,…}, the video segment at time slot k start at t_k and take T_k seconds to finish downloading. Let $a_m(t_k)$ is the indicator if the segment with quality m is downloaded at time slot t_k :

$$a_m(t_k) = \begin{cases} 1\ if\ the\ video\ segment\ with\ quality\ m\ is\ download\ at\ t_k \\ 0\ otherwise \end{cases}$$

Consider a video player have a buffer consist of Q chunks of pre downloaded video segment, each segment have the length of p seconds of the video content:

$$Q(t_{k+1}) = \max\left[Q\left(t_k - \frac{T_k}{p}\right), 0\right] + \sum_{m=1}^{M} a_m(t_k)$$

At the start of a time slot t_k , BOLA will make a decision on which video quality m to download in order to maximize the user experience. The action mentioned above requires the algorithm to solve the following equation:

$$\text{Maximize } \frac{\sum_{m=1}^{M} a_m(t_k)(Vv_m + V\gamma p - Q(t_k)}{\sum_{m=1}^{M} a_m(t_k)S_m}$$

$$\text{Subject to } \sum_{m=1}^{M} a_m(t_k) \leq 1,\ a_m(t_k) = \{0,1\}$$

Where V and γ are control parameter that can be set manually by human, S_m is the average bitrate level of the video segments belong to quality m and V_m is the assigned utility points reward to the algorithm after downloading the video segment with quality m

BOLA has been proven to be very resistance to sudden drop in network throughput. This is partly due to buffer is a more conservative variable compare to network throughput directly which directly lead to a much more abundance buffer to absorb sudden drop. However, it also means that BOLA usually reacts slowly in situation where a sudden increase in network throughput is introduced. Another requirement in order to apply BOLA is that video player needs to have a large buffer to support the algorithm result in BOLA may not be optimal when come to situation like live record event

Dynamic rule

As comparison prior, both Throughput rule and BOLA rule have its own strength and weakness. Therefor it is natural to thinking of combining these two rules under the same ABR in order to improve on QoE. By implementing Throughput when the buffer is low or empty and BOLA when the buffer is sufficient, the new design ABR can be benefited by both the stability of BOLA and the fast reaction rate of the Throughput. Dynamic rule is currently the default ABR of dash.js and is used widely (Spiteri, Sitaraman, & Sparacio, From Theory to Practice: Improving Bitrate Adaptation in the, 2018)

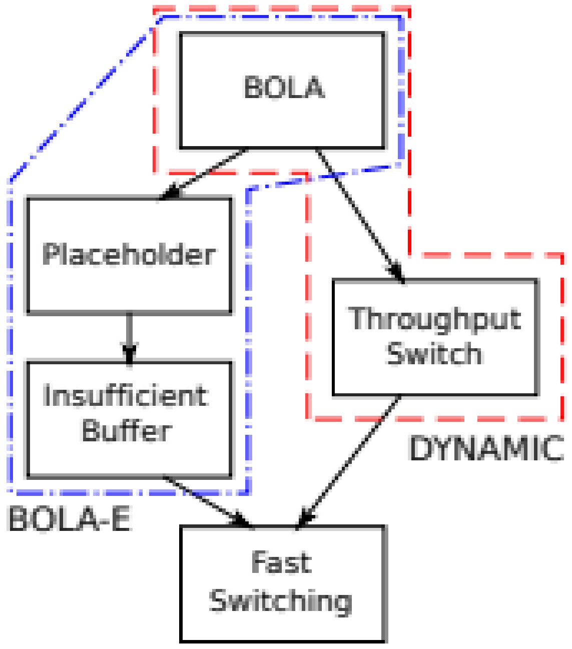


Figure 2.1: Various options can be used in Dash.js

2.2. Neural network in machine learning

For the most part of machine learning have been getting the attention of both scientific researchers and big service providers, neural network has been an integral part for the capability to process and approximate huge amount of data. As more and more algorithms and techniques are discovered and develop, so is the architecture of neural network is transform. Although this thesis doesn't look deep into this matter, a basic and throughout knowledge of such part is essential to support the algorithm to it best potential

Neural network

In order to understand the neural network, we should first have a look at it basic component the neuron. A neuron is simply a processing unit do some operation (usually a linear operation in the form ax+b) to transform some amount of inputs into different output. A neuron usually comprise of weights, biases and an activation function

- Weights are the coefficients that used to multiply directly to the input denote as w_i. Each input will corresponding to one particular weights
- Biases are the addition to the operation done by the neuron usually denote as b_i. Similar to how each input corresponding to a weights, each output is corresponding to its own bias
- An activation function is the final step before the data is ready to output after processed by weights and bias. It is a simple differentiable equation that has a predictable

form. Depend on the desire data type, properties, … on the output, there are different activation function that can be utilized

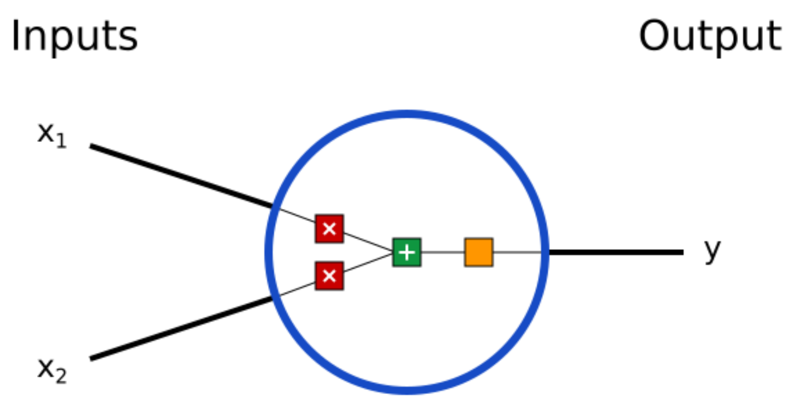


Figure 1.2: A simple two input neuron

Combining a lot of neuron together will result in an neuron network. With a wide range of computation and combination of weights, bias and activation function, a neural network can detect and learn features that represent the input environment and roughly estimate various output. In this thesis, a neural network is crucial to implement a fucntion approximator and policy approximator that we will discuss later

Loss and optimizer

Training a neural network is the process of mapping from the a set of input into the set of output. In order to complete the task, we need a way to evaluate the how well the neural network able to map to the output and a method to alternate network attribute based on the result given in prior comparision. That is the loss and the optimizer function

- The cost or loss function has an essential role in analyse and summarize the model into a single number so that an improvement made to the number is equivalent to an improvement to the model in overall. Two most crucial loss function in machine learning is cross-entropy loss and mean-square error
- Optimizers are algorithms or methods used to change the attributes of your neural network such as weights and learning rate in order to reduce the losses. The most famous Optimization algorithm is Gradient Descent which is an algorithm depend on the first order derivate of a loss function. It will try to reach a minima determine a direction in which the weight should be changing. Through backward propagation, the loss is transferred from one layer to another and the model's parameters also known as weights are modified depending on the losses so that the loss can be minimized

2.3. Reinforcement learning overview

Finite Markov Decision process

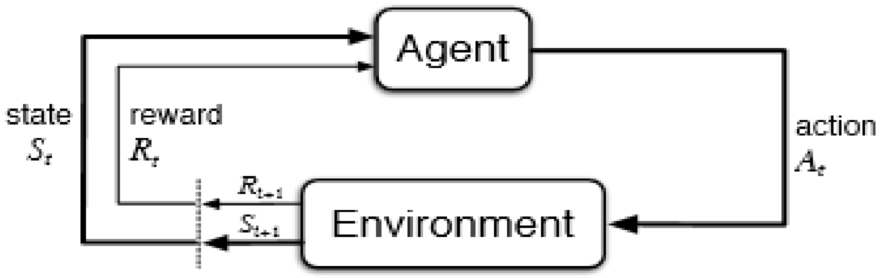


Figure 2.3: Markov Decision process model

MDP is a mathematical formalized model of sequential events. In essence, MDP is the simplest form of learning via interaction to achieve a goal. MDP composes of two main elements: Agent and Environment. These two elements interact with each other in a continually state:

Where agent is the basic of which decision should be made (action A) while the environment is responsible for process action and provide a presentation of the various condition of current environment factors (state S) and a reward R which is mathematical number which provide a stimulus for the agent the reach for the desired goal

The return or long-term reward is the ultimate goal of the agent seeks to maximize. This return can be described as a series of multiplication between the problem reward and discount factor. Discount factor is a hyper-parameter and can be adjust for individual problem which illustrate the important of the immediate reward with respect to delay reward.

Discount factor also useful for situation where the end of the interaction between agent and environment is not well define. If the reward of next state is only affected by some limited amount of current action, the discount factor can be adjust so that reward further into the past gradually diminish to zero as they are not important to the learning process anymore

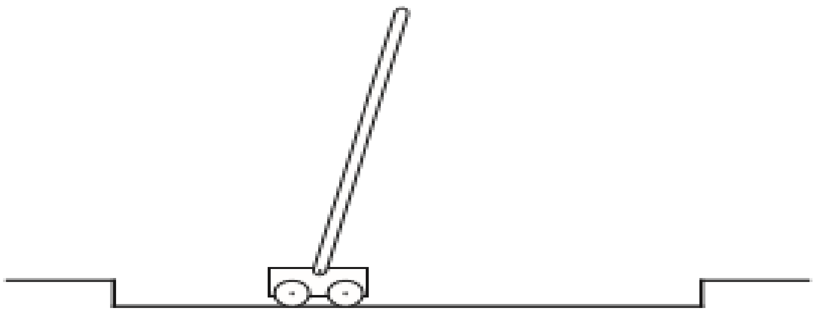


Figure 2.2: Cartpole problem

One of the classical problems that usually represented using a MDP is the Cartpole problem (also call the reverse pendulum problem) where a pole must be balance on top of a sliding cart. In this example we can clearly see how different part of MDP is expressed: the main goal is to keep the pole balance as long as possible not temporary and previous decision can affect the state that result instead of the action that just been made in the prior step. This problem also a continuous problem but the further back the action is made the less likely that it affects the result state

The Bellman equation

In order to evaluate the action and learning from environment interaction, the agent utilized a policy and value function. Policy is the instruction of how agent supposes to react to the given state, value function on the other hand is the estimation of how good the action will turn out according to agent experience, essentially a simplified version of the environment

In a typical reinforcement learning problem, there are many reasons why a complete model of the environment cannot be obtained. Usually, the environment is too complex to capture the full interaction between components. Another situation is that the amount of state space is too extreme that capturing all will be expensive in term of computer resources. Also the state-value function may actually be continuous in which case it also impossible too fully capture the interaction that may happen in the environment

Formally, a policy is illustrated as a mapping between the environment state and probabilities of pick actions (denoted as $\pi(a|s)$ for probabilities of picking action a in state s). Value function, on the other hand, can be represent as state-value function (denoted as $v(s)$ for the value of the state s if using the current policy) or state-action value function (denoted as $q(s, a)$ for the value of the state s if the action a is taken) both interpretation is used widely in reinforcement learning problem

$$v_\pi(s) = E_\pi[G_t|S_t = s] = E_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\middle|S_t = s\right], for\ all\ s\epsilon S \qquad (1)$$

$$q_\pi(s,a) = E_\pi[G_t|S_t = s, A_t = a] = E_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\middle|S_t = s, A_t = a\right],$$

$$for\ all\ s \in S, a \in A_t \qquad (2)$$

One fundamental property regardless of which form of value function is used is that they all satisfy a recursive relationship with its successor value for all policy and state

$$v_\pi(s) = E_\pi[G_t|S_t = s]$$

$$v_\pi(s) = E_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \qquad (3)$$

$$v_\pi(s) = \textstyle\sum_a \pi(a|s)\sum_{s'}\sum_r p(s',r|s,a)[r + \gamma E_\pi[G_{t+1}|S_{t+1} = s']] \qquad (4)$$

$$v_\pi(s) = \textstyle\sum_a \pi(a|s)\sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s'), for\ all\ s \in S, \qquad (5)$$

This is called the Bellman equation, and the objective of any reinforcement learning problem is to maximize this function. There are two options in which the Bellman equation can be deployed into a reinforcement learning agent.

- Monte Carlo approach: using the sum of all return as the guideline for the agent to improve. Mathematically speaking, Monte Carlo approach is equivalent to the (3) equation in the Bellman equation. This approach is proven to be quite robust and have higher capability of converging
- Differential Learning approach: the equivalent of the (5) equation. This approach using the immediate estimation of the value of the consecutive state in order to update the current state. This technique is considered as bootstrap (learning base on itself). Although Differential Learning make the algorithm more vulnerable to starting value and make it a little bit problematic when it comes to convergent property, it helps with various problem of consecutive control where the end of the episode either not exist or hard to define as this approach doesn't relied on the terminal state

Value approximation

Classical reinforcement learning problems usually have a relative manageable and discretize value function. In those problems, it is clear that a simple solution such as a look up table is sufficient to have them solved. In fact, tabular solutions are considered to be very accurate and precise. However, the problem here is how theoretical such situation happens in real life. In most cases, real life reinforcement problems have very complicated value function. Some example of a real world problem state space that reinforcement learning is used to deal with:

- Go, a traditional Chinese game, it is estimated that go have about 10^{170}different states. With such a massive state space it is no way that the a computer hardware would be capable to save an entry for each of them
- Cartpole problem, despite being a classical problem, also present the problem of continuous state space. Although the number of environment features in this problem is small, it is presented to the agent as a list of real number which make table entries a lot more complicated

Furthermore, even in problem where saving all the state-value pair is possible, it may still be to slow for the agent to learn all of those using the tabular method

Therefore, it is safe to say that basic problems have an abundance of complication issue translate to the real world model, especially if tabular method is applied. Thus, a more general method is used called value approximation. In this approach, instead of a table that match state with value, a function is applied and have to approximate the value of the current state given environment feature as it input (called a feature vector). In fact, value approximation argues that the value function can be represented as functional form with a weight vector $w \in$ R^d. The function itself (written as $\hat{v}(s)$) can take many forms but the most useful one and also the interest of this thesis is the representation of value function as a multi-layer neural network.

It is natural to think of the update of the value function in reinforcement learning, regardless of the method of interest, as an example of input-output interaction that is desired to be archive by a neural network. Therefore, we can consider value approximation as a supervised learning problem.

$\Delta \mathrm{w} = \alpha(G_t - \hat{v}(S_t, w))\nabla_w \hat{v}(S_t, w)$for Monte-Carlo approach

$\Delta \mathrm{w} = \alpha(R_{t+1} + \gamma\hat{v}(S_{t+1}, w) - \hat{v}(S_t, w))\nabla_w \hat{v}(S_t, w)$ for Temporal Different approach

An obvious issue with the way of seeing the approximating the state-value function as a supervised learning problem is the lack of supervisor. This can be partially offset by the use of prediction methods as the reference for updating the weight as shown in the formula above. Although this fact is a risk factor in the capability of the estimator to converge

Using function approximation comes with its own set of advantages and disadvantages:

- Advantages:
  - Updating weight instead of specific value in the table allows the agent to generalize and pick the appropriate action even if the state have never been seen before
  - The use of function weight w allow the agent to save memory for problem with large state space (there are less weight comparing to the number of state)
- Disadvantages:
  - Updating from one state-value pair may end up affecting the result of other state-value pair due to the nature of value approximation. This behaviour could lead to some potential issues in convergent
  - The used of prediction method as mentioned, raise some convergent issue. This is especially true in the case of Temporal Difference method as this evaluation method heavily based on bootstrapping data instead of an actual objective function

The disadvantages in convergent in value approximation has been illustrated in Baird's counter example:

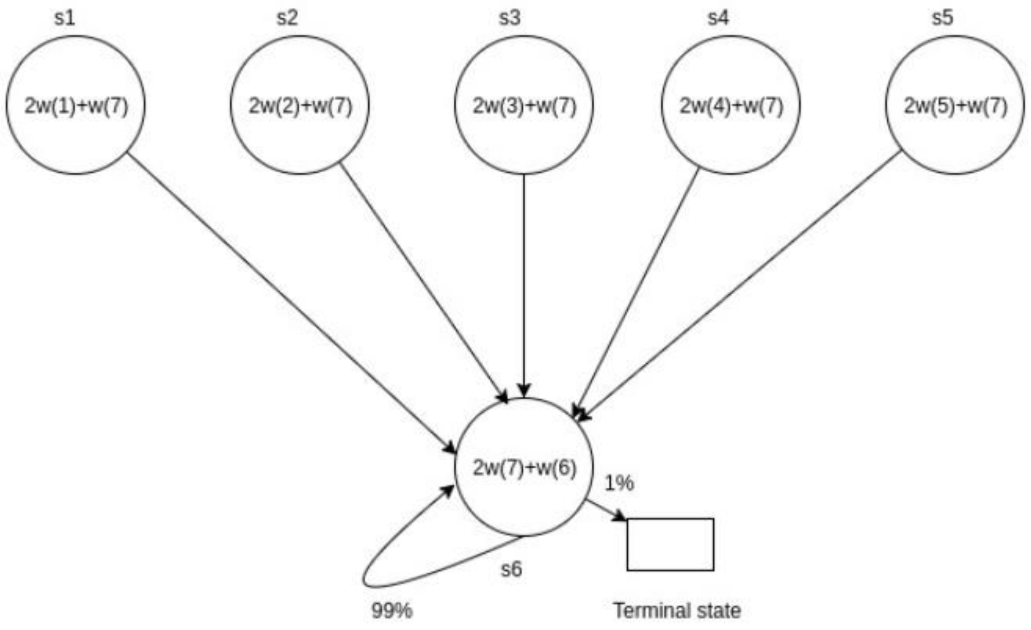
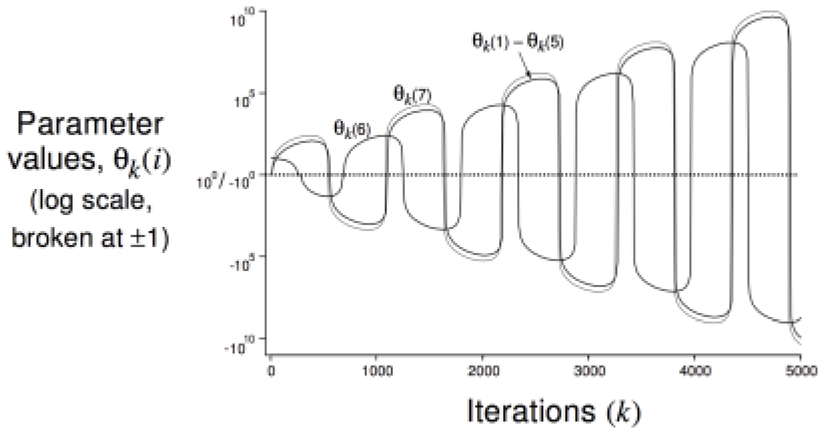


Figure 2.3: Baird's counter example MDP visualized

The MDP shown in the figure consists of 6 non-terminal states and 1 terminal state. It is an episodic MDP. Episodes start in one of the five upper states (picked uniformly at random), and proceed immediately to the lower state. When the agent takes (the only available) action in the lower state, it remains in the same state with probability 99%; with probability 1% it goes to the terminal state (and the episode ends). Take the discount factor γ to be 0.99. All rewards are zero rewards. By default the value of the terminal state is also taken to be zero.



Model-free control

In the previous section, we have walked through the method of evaluating a policy. However, knowing the value alone does not solve the problem of optimization that usually presented to a reinforcement learning agent. Instead, an extra step needs to be taken, rather than evaluating a policy, the agent real job usually to optimize it. In order to approach this, there are two viable options that the algorithm can be design: on-policy and off-policy

- On-policy approach uses the learning policy that in consideration as the main guide line for the agent to discover and improve state-value function approximation. Due to the heavy bias toward the current policy as the main guide line to explore the environment, additional exploration policy is usually applied to avoid local optimum which we will discuss later. Some classical RL method that is on-policy: Monte-Carlo exploration start, SARSA
- Off-policy approach on the other hand deviates away from the learning policy to explore the environment and improve state-value function while try to improve the learning policy. With that approach it is guarantee to maintain some level of exploration without any additional help. Some classical RL method that is on-policy: Monte-Carlo important sampling, Q-learning

Both are valid choice when comes to building and agent. On-policy allow fast convergent and have higher data efficiency while off-policy offer a simpler implementation with better final convergent, more resistance against local optimum. This thesis will mainly focus on On-policy approach

Exploration and exploitation

Exploration and exploitation are two most important features of a RL agent. Exploration is the capability to discover the environment, deviate from the current path in order to discover a better solution while sacrificing temporary advantage of consistency of a well-known path. While exploitation is the capability to use and rely on already learned information and make the best option to optimize the value of return given a specific situation. Without exploration the agent can never learn any information from the environment while without exploitation the agent will act extremely inconsistent and suffer major convergent issues. Therefore, a large part of reinforcement learning research is dedicated to study an optimal trade-off between these two properties. In this thesis, for the sake of simplicity, we will only focus on two major approaches: decay epsilon and entropy maximize framework

- Decay epsilon is the technique of using a variable (epsilon) to determine the agent behavior. By applying epsilon, the agent has a small chance that it will act completely random (exploring the environment) while still be able to maintain the ability to act according to the collected knowledge (exploiting) most of the time. The decay factor will determine the rate of which epsilon reduce (less exploring) as more and more iterations have passed
- Entropy maximize framework, in the other hand, will encourage the agent to take action in which has higher entropy. This technique only viable in situation where the policy can be represented as a percentage distribution represent the way the agent pick action according to its knowledge. Due to the nature of the entropy number $H(x) = -\sum P(x) * \log x$, the agent can be encourage to explore by having entropy added into the return. Another variable call temperature variable will determine how much exploration is required before the agent can actually evaluate it action as well as acting as a normalize coefficient

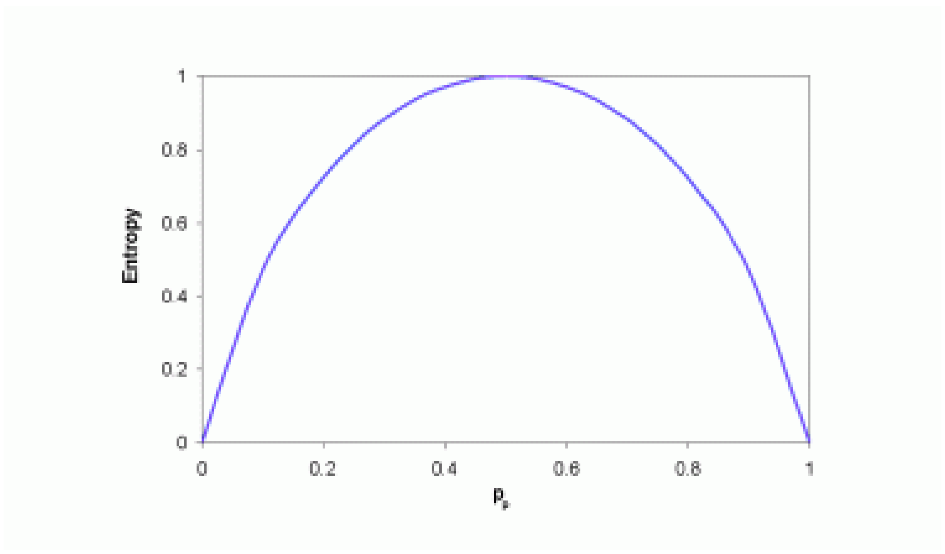


Figure 2.4: Entropy function. The closer P is to 0 and 1 (deterministic) the smaller the entropy

Reinforcement learning algorithm

In summary, there is a huge array of properties and technique that can be applied when approaching a particular reinforcement learning problem. Each combination gives an algorithm with its own set of benefits and hindrance and can be suitable or not depend on the problem. Here is a quick summary on method that obtained a lot of attention in recent reinforcement learning algorithm:

- Policy Gradient Methods: A common technique that is used throughout machine learning general is stochastic gradient decent which is an algorithm that allow AI agent in general to move in the direction that is the most optimal in order to converge. Combining this approach into reinforcement learning framework will result in a family of algorithm call policy gradient methods. These methods capitalizes on the fact that the value function is not the target of most RL problem and only focus on changing the policy in the way that offer the most long term value (or most advantage) by incentivize the agent to take aggressive changes in the direction which have the most benefit. This method is at the center of many major breakthroughs in Google OpenAI project as it is compatible with entropy maximize framework as well as simple enough to implement while still giving very impressive result. This family of methods includes Vanilla policy gradient decent, Proximal Policy Optimization (PPO) (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017)

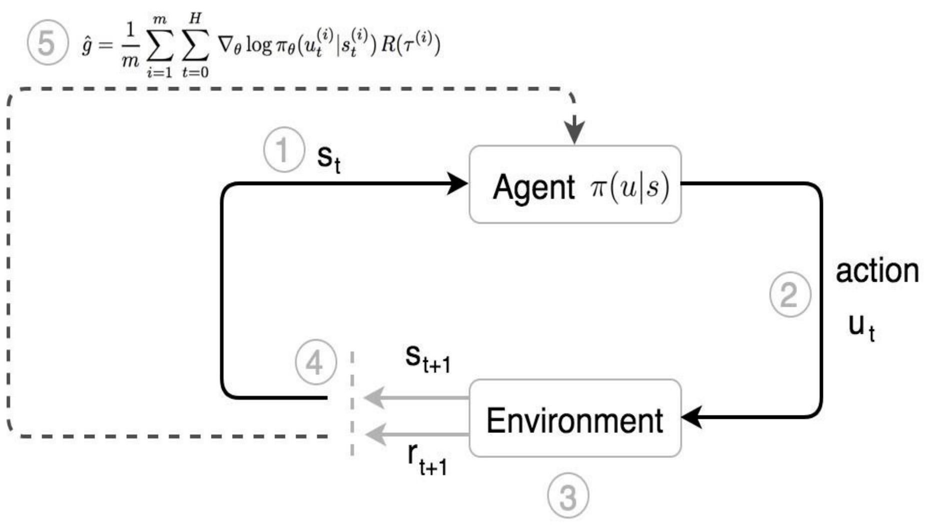
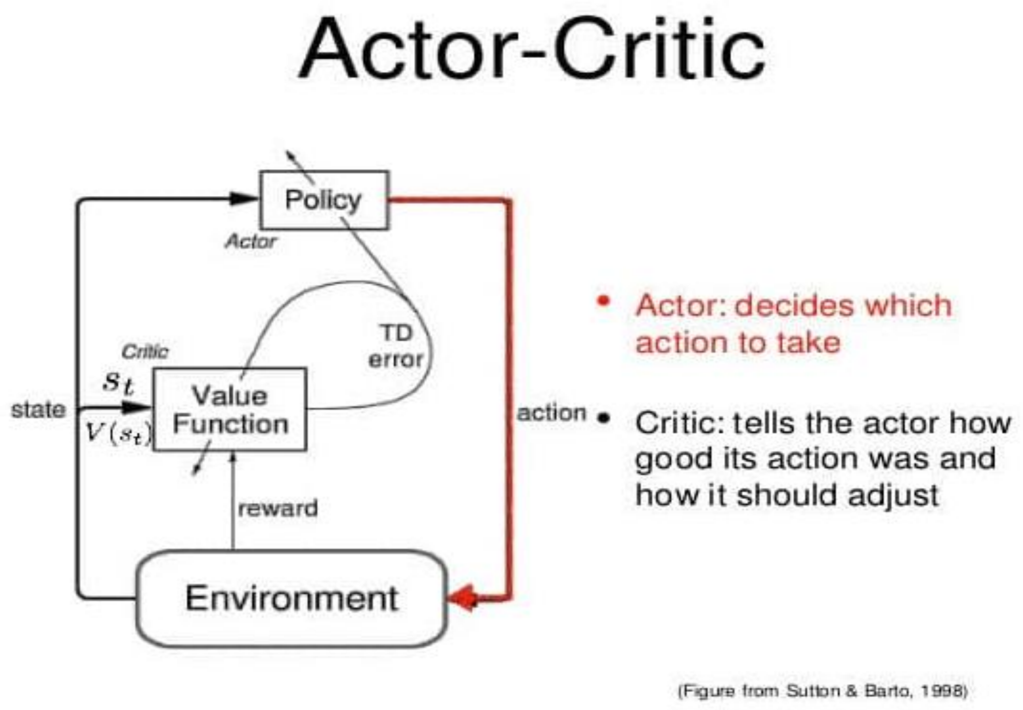

5
$\hat{g} = \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{H}\nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})\, R(\tau^{(i)})$
1
s_t
Agent $\pi(u|s)$
2
action
u_t
4
s_{t+1}
Environment
r_{t+1}
3

Figure 2.5: Policy gradient method mechanism

- Actor-Critic Methods (Konda & Tsitsiklis, 2000): These methods are among the most recognizable RL algorithm. These methods a combination of DQN and a typical policy gradient method where the DQN is called the critic and the policy gradient agent is called the actor. The idea is to utilize the DQN as an expectation for the policy gradient method instead of directly evaluate on the environment with the return, result in a more stable method than Policy Gradient Methods but still have a decent convergent property and resistance to bias. Within this family of method there also A2C (Advantages Actor-Critic) and A3C (Asynchronous Advantages Actor-Critic) which are both include many improvement in order to reduce train time as well as improve data efficiency



Actor-Critic

- Actor: decides which action to take
- Critic: tells the actor how good its action was and how it should adjust

(Figure from Sutton & Barto, 1998)

Figure 2.6: Actor-Critic mechanism

CHAPTER 3

Methodology

3.1. Neuron network construction

In order to deploy a neuron network in this thesis, a library name Tensorflow will be implemented. TensorFlow is an open-source software library. TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains

TensorFlow provides multiple APIs (Application Programming Interfaces). These can be classified into 2 major categories:

Low level API:

- Complete programming control
- Recommended for machine learning researchers
- Provides fine levels of control over the models

TensorFlow Core is the low level API of TensorFlow.

High level API:

- Built on top of TensorFlow Core
- Easier to learn and use than TensorFlow Core
- Make repetitive tasks easier and more consistent between different users

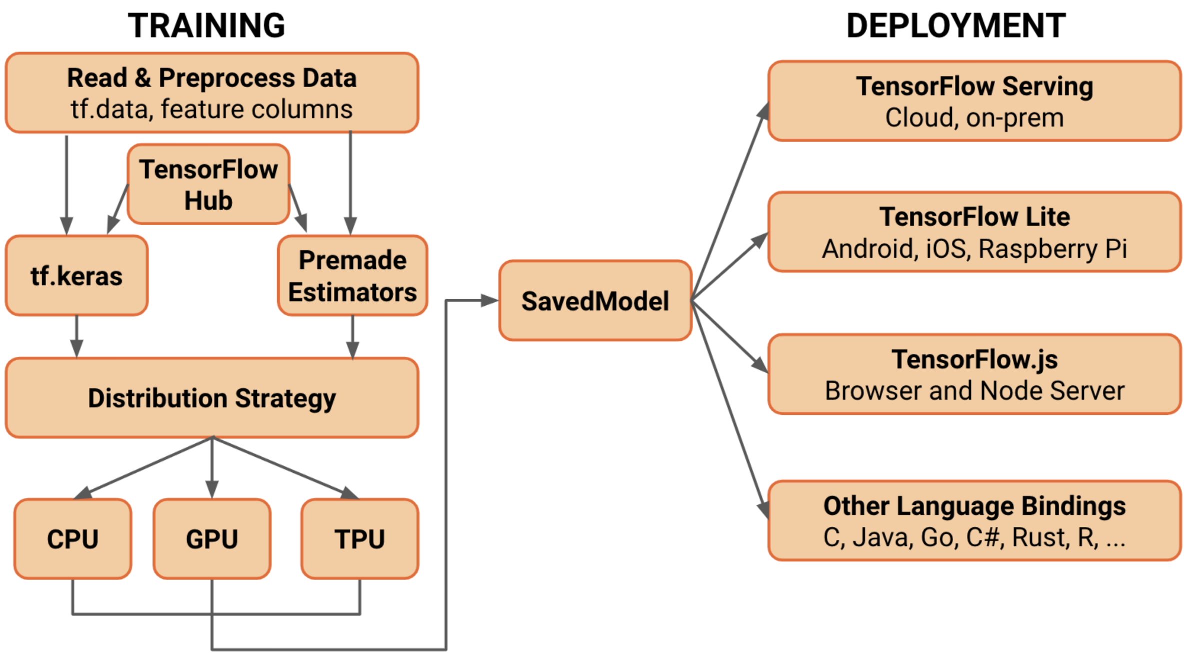


Figure 3.1: Tensorflow 2 simplified architecture

tf.keras is an example of a high level API. It is the most welcomed improvement on Google update into Tensorflow 2. It is easier for new user to engage with thanks to friendly design combine with intuitive high-level API. For researcher, on the other hand, the framework is generalized enough while still allow user to quickly deploy there idea into the real life problem

Another advantage of Keras is its great flexibility build a top of a solid, well-structured modular building block allows user to express a wide range of creativity.

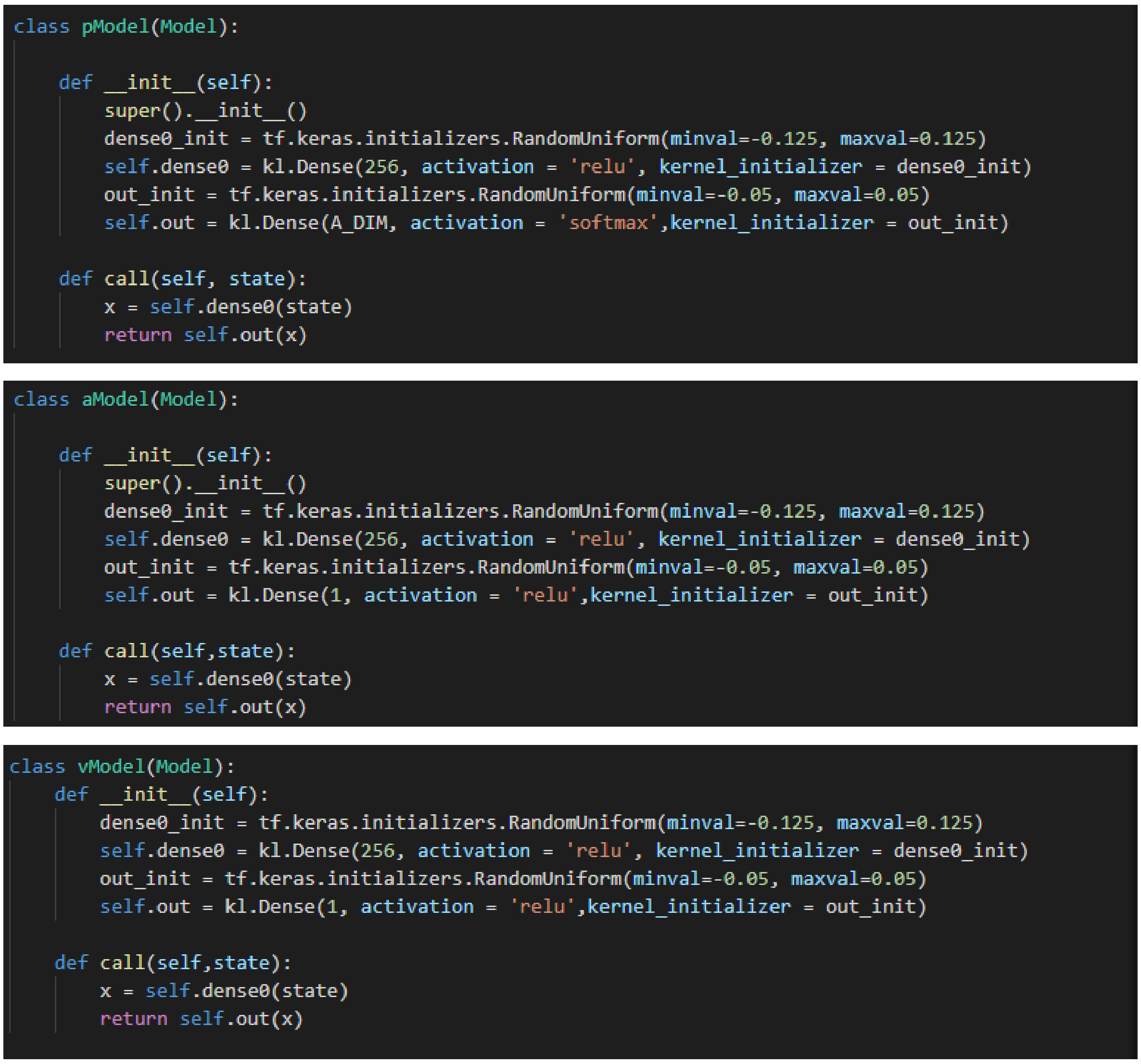
In this thesis, there are two types of neural network needed two construct various algorithm mentioned.

The first type is a neural network for value approximation. This type of neural network is fairly standard as it is pretty much a neural network used for regression. In theory, we need an output for every action possible in our problem and that is what will be used for DQN. A3C on the other hand, using advantageous function specifically, not value function, which only require an overall average estimate of the current policy therefor only one linear output is required.

The second type of neural network is for the policy approximation. This type of network utilizes a probability distribution to represent a policy. Therefore we will have the same number of output as the number of action. To deal with probability it is necessary to use softmax activation function

The use of a dense network is necessary to detect various “features” that is helpful to determine the final value. However, as the Pensieve [Mao, Netravali, Alizadeh, 2017] author pointed out, an increase in the number of dense layer does not necessary translate to an increase in performace. Therefor in this thesis, we will stick on a sample neural network with one dense layer using relu activation function (a kind of linear activation function)

Kernel initialization is a feature of keras which is used to initialize weight for a keras model. According to research [Salimans Kingma]



```python
class pModel(Model):

    def __init__(self):
        super().__init__()
        dense0_init = tf.keras.initializers.RandomUniform(minval=-0.125, maxval=0.125)
        self.dense0 = kl.Dense(256, activation = 'relu', kernel_initializer = dense0_init)
        out_init = tf.keras.initializers.RandomUniform(minval=-0.05, maxval=0.05)
        self.out = kl.Dense(A_DIM, activation = 'softmax',kernel_initializer = out_init)

    def call(self, state):
        x = self.dense0(state)
        return self.out(x)
```

```python
class aModel(Model):

    def __init__(self):
        super().__init__()
        dense0_init = tf.keras.initializers.RandomUniform(minval=-0.125, maxval=0.125)
        self.dense0 = kl.Dense(256, activation = 'relu', kernel_initializer = dense0_init)
        out_init = tf.keras.initializers.RandomUniform(minval=-0.05, maxval=0.05)
        self.out = kl.Dense(1, activation = 'relu',kernel_initializer = out_init)

    def call(self,state):
        x = self.dense0(state)
        return self.out(x)
```

```python
class vModel(Model):
    def __init__(self):
        dense0_init = tf.keras.initializers.RandomUniform(minval=-0.125, maxval=0.125)
        self.dense0 = kl.Dense(256, activation = 'relu', kernel_initializer = dense0_init)
        out_init = tf.keras.initializers.RandomUniform(minval=-0.05, maxval=0.05)
        self.out = kl.Dense(1, activation = 'relu',kernel_initializer = out_init)

    def call(self,state):
        x = self.dense0(state)
        return self.out(x)
```

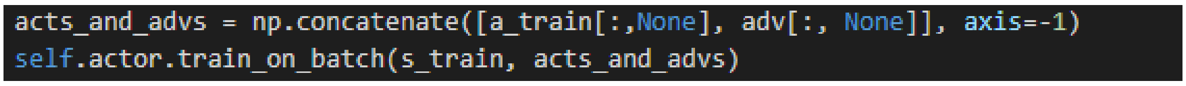
Figure 3.2: Neural network for policy approximation

3.2. Loss and optimize function

There are two kinds of loss functions that need to be used in this thesis. On one hand, regression loss use for both value function and advantage function. For this task a simple mean square error loss function can be applied. On the other hand, policy approximation function requires a non-conventional approach to implement. Our policy approximation has the output of a typical categorical problem, however instead of a label telling the result is correct our not, we need to incorporate reward into the loss function as different reward have different effect on policy approximation function. Therefore, instead of the conventional categorical cross-entropy loss function like in a categorizing problem, the policy network utilized a weighted categorical cross-entropy loss:

$$loss = \nabla_\theta log\pi_\theta(s,a) * A^{\pi_\theta}(s,a)$$

The function requires two parameters to be passed along with the state in order to calculate which Tensorflow unfortunately does not support. However, thanks to a clever solution suggested by Tensorflow team in one of their blogs which is to pass the value as a dictionary, we can carry on to implementation of algorithm



```
acts_and_advs = np.concatenate([a_train[:,None], adv[:, None]], axis=-1)
self.actor.train_on_batch(s_train, acts_and_advs)
```

Figure 3.3: Recommended work around by Tensorflow team

This is also where actor-critic algorithms differ from the pure policy gradient method. Where in policy gradient method the advantage $A^{\pi_\theta}(s,a)$ is substitute directly with the value obtained from Monte-Carlo method or Temporal Different method, actor-critic in general using the different between the value and a baseline which in this case the average or the advantage function of the current policy

If a temperature variable is used to encourage the policy gradient and actor-critic gradient to explore more, policy's loss function is the appropriate place to add

In term of optimizer, Keras api offer a wide range of choice, but a simple stochastic gradient decent optimizer is enough to observe changes in the training process. It is also observe that for actor-critic algorithm, the learning rate of the advantageous function should be less than the learning rate of the policy function for reason will be further discuss below

A simple method of decoupling data is also used which is called experience replace. This method using a memory to record the data gather while agent is running in the simulator, only sample randomly a small amount of data and present them unordered in the training process

3.3. Implication of RL framework into rate adaption

In order for the computer to make a well-informed decision on which bitrate to choose from, there are a couple things need to be concerned.

Firstly, it is important to supply the learning agent which appropriate information. The agent need to be able tell the network condition and make a prediction on how the internet will changes in the future. In order to accomplish this task it is important to provide some amount of past network data. In our case it is decided that average bitrate of the last 7 segments as we want the agent have the most complete picture possible of the environment while still maintain a fast training and predicting agent. Too little segments being accounted for and the prediction will be too incomplete to be used while too much segments lead to a reduction in efficiency of the network.

Secondly, as proven by BOLA, the buffer level is an important piece of information to make a decision. A high level of buffer will allow more aggressive decisions to be made as the risk of encounter re-buffering is smaller as the buffer grows. Furthermore, a full buffer will lead to a delay in the download activity which is definitely will reduce the overall efficiency of the adaption method.

Thirdly, as shown in the dataset section, video segment size vary quite a bit depend on the information contain in each segment. Therefore, it is also important for the agent to have access to this information.

Finally, smoothness is also an attribute that need to be concerned about according to QoE standard. This information can be supplied to the artificial intelligent by introducing the latest action that has been made

It is expected that the agent will be capable to evaluate the stated value above and make a careful decision that can balance the above factor in order to maximize the reward which in this case is the user satisfactory which is indicated by the result discuss above

3.4. Simulation

Most reinforcement learning problems cannot be trained in real world problem. This is due to algorithm nature of data inefficiency as well as expense for such approach would be enormous. This research into DASH is also not an exception to that problem. To tackle this problem, most reinforcement learning problems is simulated on the virtual environment where data can be looked into, clean up and thousands of hours of real world training can be capture in an instance. Consider our problem, each episode includes 60 video segments of size 4s, which means that it take 4 minutes to run an episode and evaluate the data. With a simple

10000 episodes of training for our single network data, it would take almost a month to have the agent trained. This process can be reduced into just around 10 minutes with the current using network simulation. Therefore, having a good simulation is very important for any reinforcement learning research

The training environment is store at video_player.py file. This file will be responsible for simulating the download process of each video segment based on the network dataset and the video segment dataset mentioned above. There is no need to simulating the playback process due to the following reasons:

- Video segment is actually played as the order as they are downloaded
- Direct buffer calculation can be made after a segment is downloaded and the disparity between buffer and download time will reflect the video state

Dataset detail

Network dataset (D. Raca, 2018) is kindly provided in the University College Cork, Ireland’s website. This dataset capture 4G network condition in various time point using a variety of transporting network. Despite some missing values as stated by the author, most of them are relate to GPS position due to the difficulty of traveling across the city terrain. The information that we are most interesting is the network download bitrate is perfectly captured. In order to represent the network download bitrate which is a continuous quantity, the paper authors have divided data into small chunks which last 1s seconds each. Download bitrate within this period is consider to have a constant speed

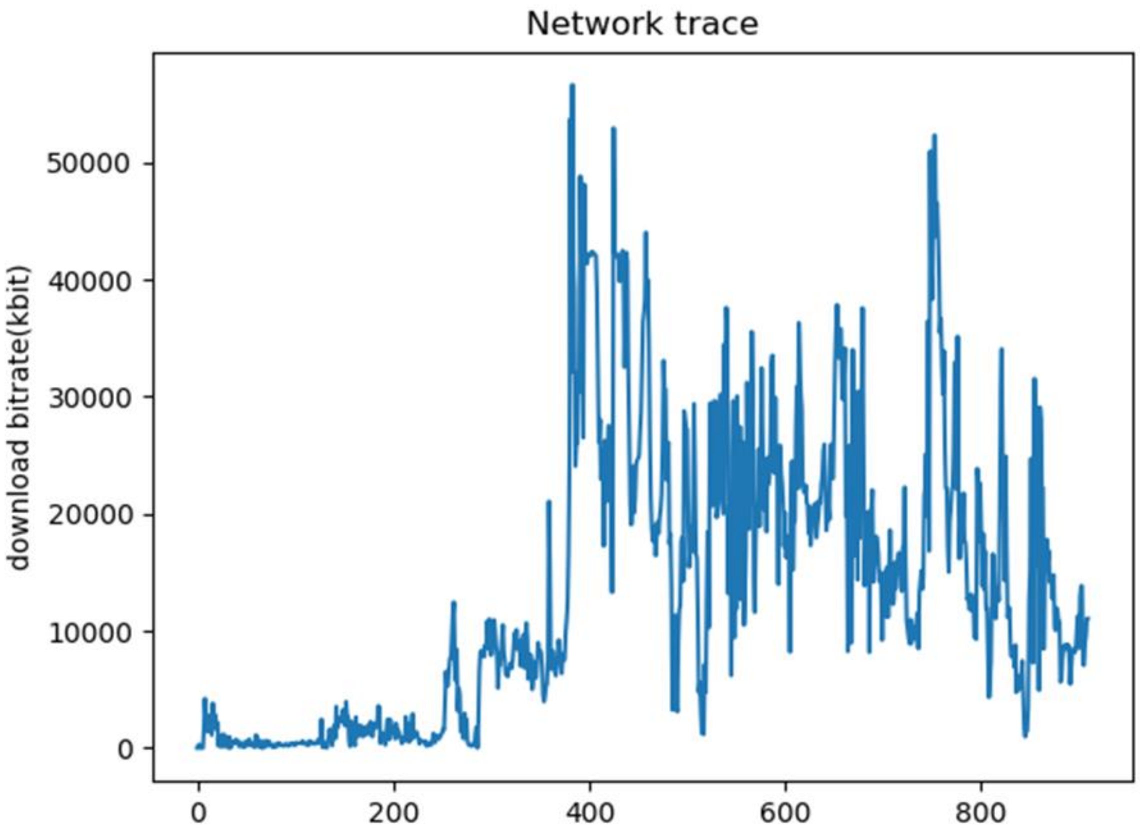


Figure 3.4: Network download bitrate

Video data used in the research are two free video use for researching dash and video taken from http://ftp.itec.aau.at/datasets/mmsys12/ database. To make sure the network is appropriate for the video as well as reducing lag this thesis will use the 4 seconds version of both video BigBuckBunny and Elephant Dream. A variety of qualities (7 in specific) are also use so that the agent has enough room to work with various network conditions. Only the segment size is used therefore there no need to download the whole video. Instead html header is used to read the content size

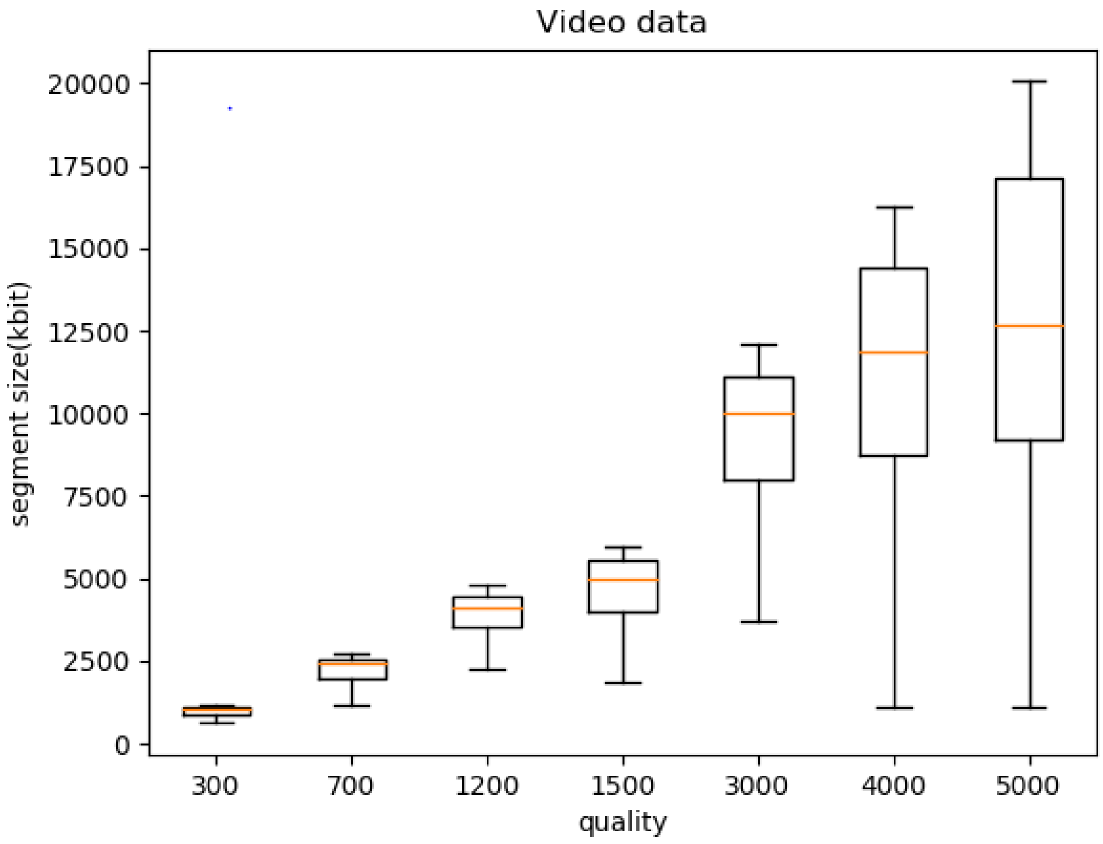


Figure 3.5: Video segment size distribution

List of variable

- ‘self.bitrate_list’: Store the currently using network data speed (kbit/s). This data is extracted from aforementioned dataset
- ‘self.video_list’: Store the currently using video’s segments’ size (kbit). This data is extracted from the aforementioned dataset
- ‘self.buffer’: Simulate the buffer available in the video player
- ‘self.buffer_thresh’: Store the maximum available space for pre storing video segment (buffer’s threshold) (in seconds). If the buffer’s threshold is reach the download process will be put into a freeze until there is more space to store the next video segment
- ‘self.net_seg_iterator’: An iterator that informs the simulation the position of the current network segment that the simulation is using
- ‘self.video_iterator’: An iterator that informs the simulation the position of the current video segment that the simulation is trying to download
- ‘self.seg_time_stamp’: A variable capture the amount of used time within a network segment
- ‘segment’: A temporary storage for the video segment that going to be download, this variable only store size and is determined by the position of ‘self.video_iterator’ and the quality chosen to be download
- ‘network’ and ‘max_throughput’: temporary variables that describe the network situation at the downloading moments. ‘network’ store the network speed while ‘max_throughput’ describe the amount of download can be done within the currently processing networking segment based on our assumption
- ‘delay’, ‘sleep_time’, ‘return_buffer’, ‘rebuf’, ‘return_segment’, ‘next_segments’, ‘terminate’, ‘remain’: variable describe the download section and will be return to the agent to study the impact of its decisions. ‘delay’ describes the delay, ‘sleep_time’ describes the freeze period, ‘return_buffer’ describes buffer size after the video segment is downloaded, ‘next_segments’ contains the size of the next video segments according to its quality used to help the agent make decision, ‘terminate’ return True if the episode have been finished, ‘remain’ return the amount of segments need to be download before the video is end

Along with the mentioned variable there are some constants that also predefined and can be adjust:

- VIDEO_CHUNCK_LEN: describes the length of a video segment in view time (in second)
- TOTAL_VIDEO_CHUNCK: describes the length of an episode (in video segments)
- BUFFER_THRESH: default buffer threshold for the simulation
- DRAIN_BUFFER_SLEEP_TIME: length of a freezing cycle when the buffer threshold is reached
- NETWORK_SEGMENT: length of a uniform speed period of network data record in previously mentioned data

Simulation description

In order to simulate the download loops the simulation utilized three stages:

- The first stage is responsible describe the freezing circle happened when the buffer is full. The ‘net_seg_iterator’ will be moved to the appropriate position after the freeze is finished in order to be further process

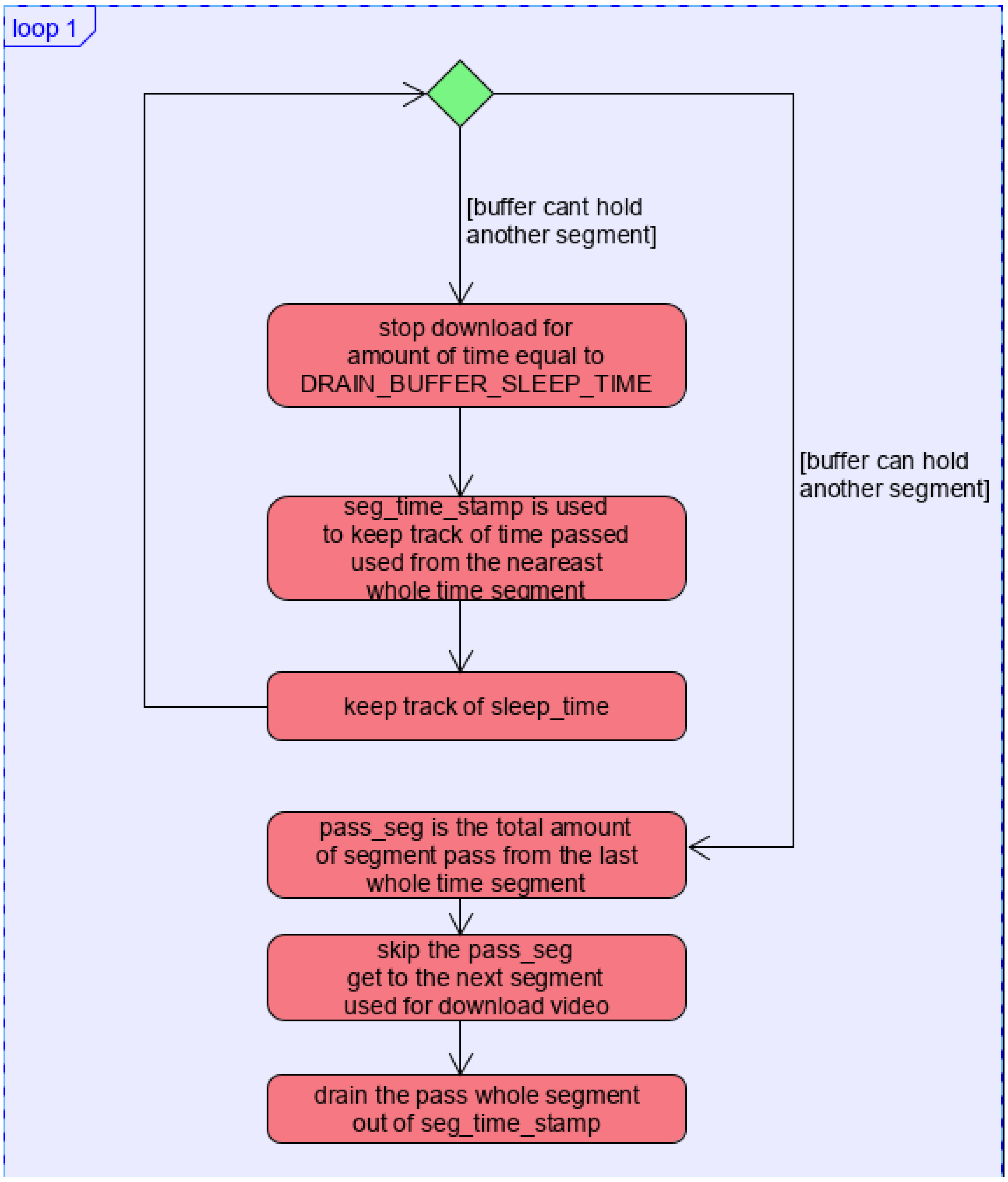

loop 1
[buffer cant hold another segment]
stop download for amount of time equal to DRAIN_BUFFER_SLEEP_TIME
seg_time_stamp is used to keep track of time passed used from the neareast whole time segment
keep track of sleep_time
[buffer can hold another segment]
pass_seg is the total amount of segment pass from the last whole time segment
skip the pass_seg get to the next segment used for download video
drain the pass whole segment out of seg_time_stamp

Figure 3.6: Freezing loop

- In the second stage, the video segment download in a repeated while loop. This loop will be presented the video segment with continuous network data segment until the total amount of network throughput has exceeded the data segment size. Any exceeded data will be represented in as NETWORK SEGMENT − self. seg time stamp and saved for the next download section. This approach is possible due to the fact that download section only affected by the network useful throughput and have no effect on the video playback

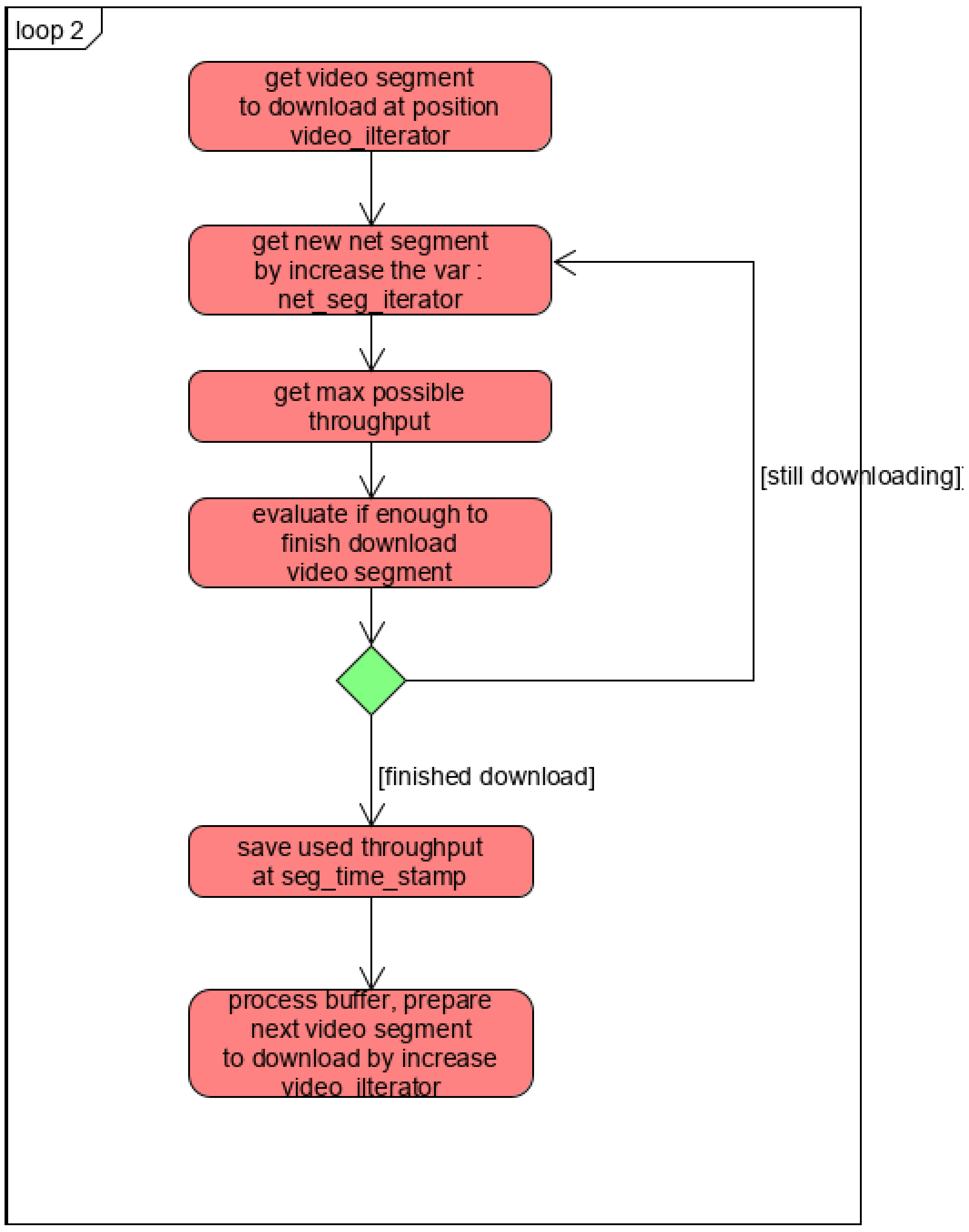

loop 2
get video segment to download at position video_ilterator
get new net segment by increase the var : net_seg_iterator
get max possible throughput
evaluate if enough to finish download video segment
[still downloading]
[finished download]
save used throughput at seg_time_stamp
process buffer, prepare next video segment to download by increase video_ilterator

Figure 3.7: Download loop

- In the third stage, the 'self.buffer' variable is processed in order to mimic the playback. This including behavior similar to record a re-buffer when the amount of delay is larger than the available buffer, refilling the buffer after the download section is finish (after the delay is processed). The return information also processed at this stage

3.5. State and reward

State and reward are the two most important aspects in deciding the effectiveness of the training environment. State describe the environment to the agent while reward predefine the goal that the agent must achieve

In term of reward, we need an agent that can make decision so that the following goal is fulfilled:

- Video does not stop or re-buffering
- Video has the highest quality possible
- Video has as little switch as possible

Where according to Pensieve [Mao, Netravali, Alizadeh, 2017] authors can be translate into

$$\text{QoE} = \sum_{n=1}^{N} q(R_n) - \mu \sum_{n=1}^{N} T_n - \sum_{n=1}^{N-1} |q(R_{n+1}) - q(R_n)|$$

With:

- $q(R_n)$: is the reward of a successful download of video segment with quality R_n, in this thesis this is simply done by rescaling R_n ‘s quality;
- T_n: buffer time due to downloading segment R_n
- μ: penalty for each delay

Consider input state, it is clear that most details are not important. In order for an agent to make informative decision it should know:

- Network speed when downloading past segment
- Current buffer size
- The amount of segment the agent still need to download
- The next available choices
- Agent last action

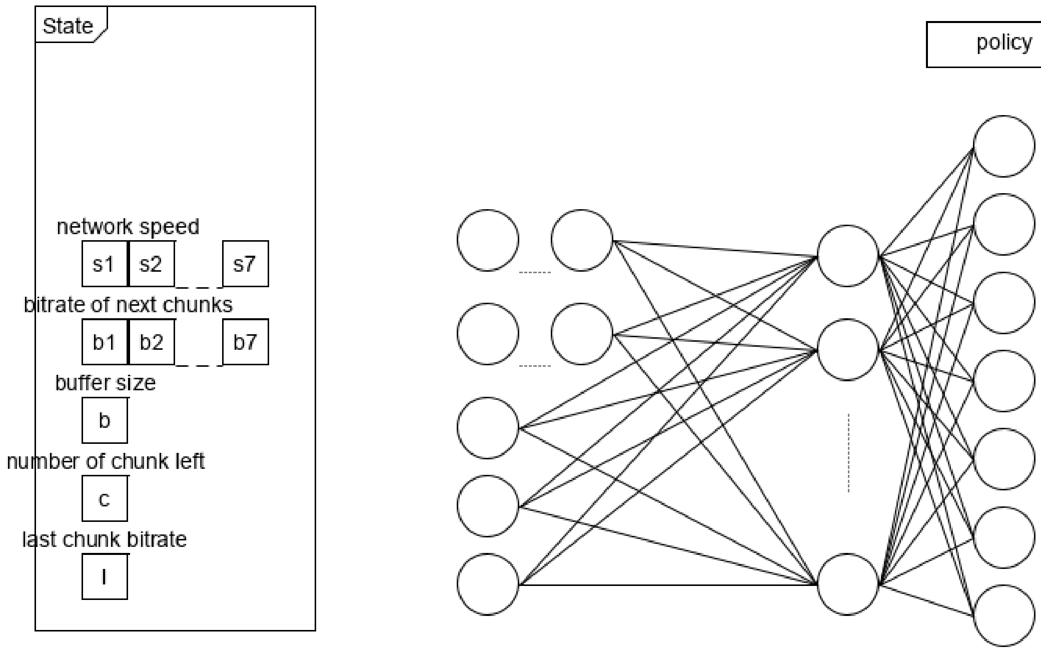


Figure 3.8: State and network visualization

Due to time constrain all the data mentioned above is only roughly scale between 1 and 10 and is based largely on the format of Pensieve. A future study into the various problems of data scaling can be made to create a more precise state and reward description

CHAPTER 4: EXPERIMENT RESULT

4.1. Training strategy

Due to the variety of dataset provided, in this thesis I decided to train on one network segment and do testing on a different dataset. This strategy will guarantee that the agent is not over fit to the training data. The video used will also be different, training data going to be BigBuckBunny video while Elephant Dream will be used as testing data

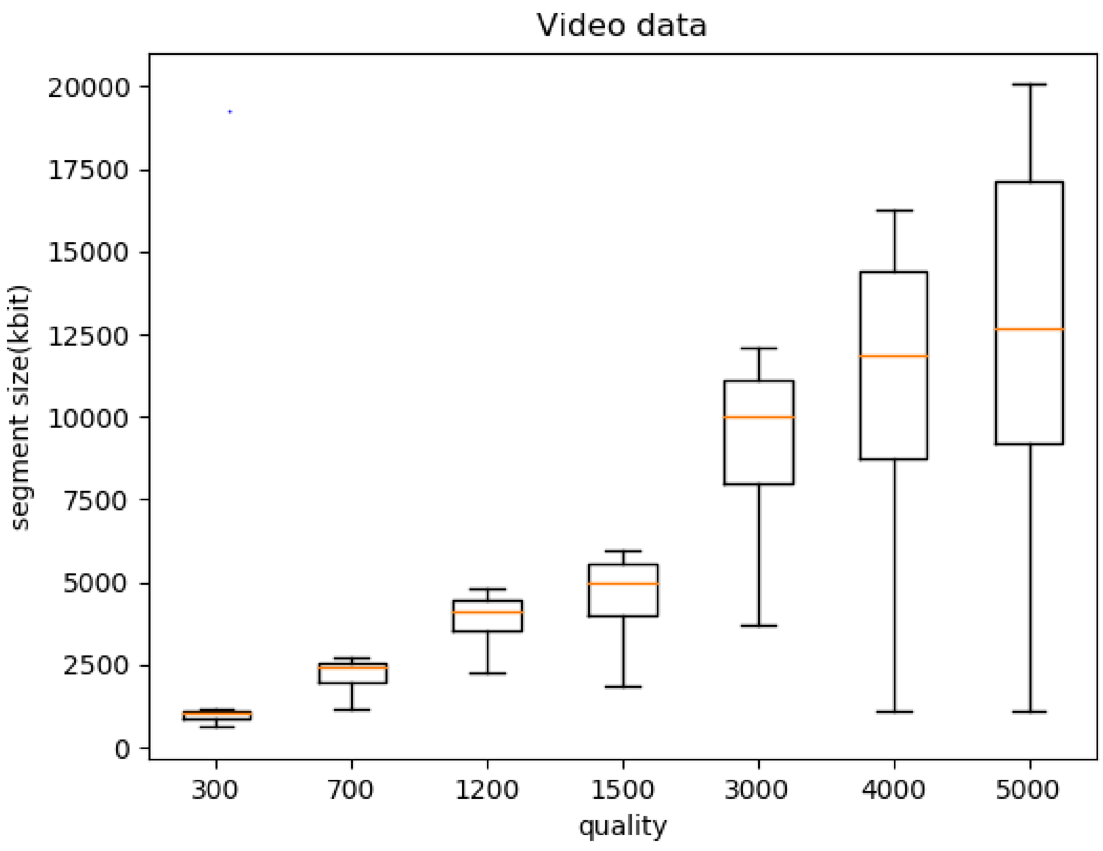


Figure 4.1: BigBuckBunny chunk size distribution

4.2. Experiment result and evaluation

This thesis used two reinforcement learning algorithm Policy Gradient and Advantages Actor Critic. Policy Gradient will use a learning rate of 0.1 with the exploration rate ε of 0.05 (5% of the time the agent will try to make a random decision instead of an informative one). Advantages Actor Critic uses a learning rate of 0.1 for actor and a learning rate of 0.01 for critic. Both algorithms will be train 10000 episode of 4 minutes worth of video and here is the result

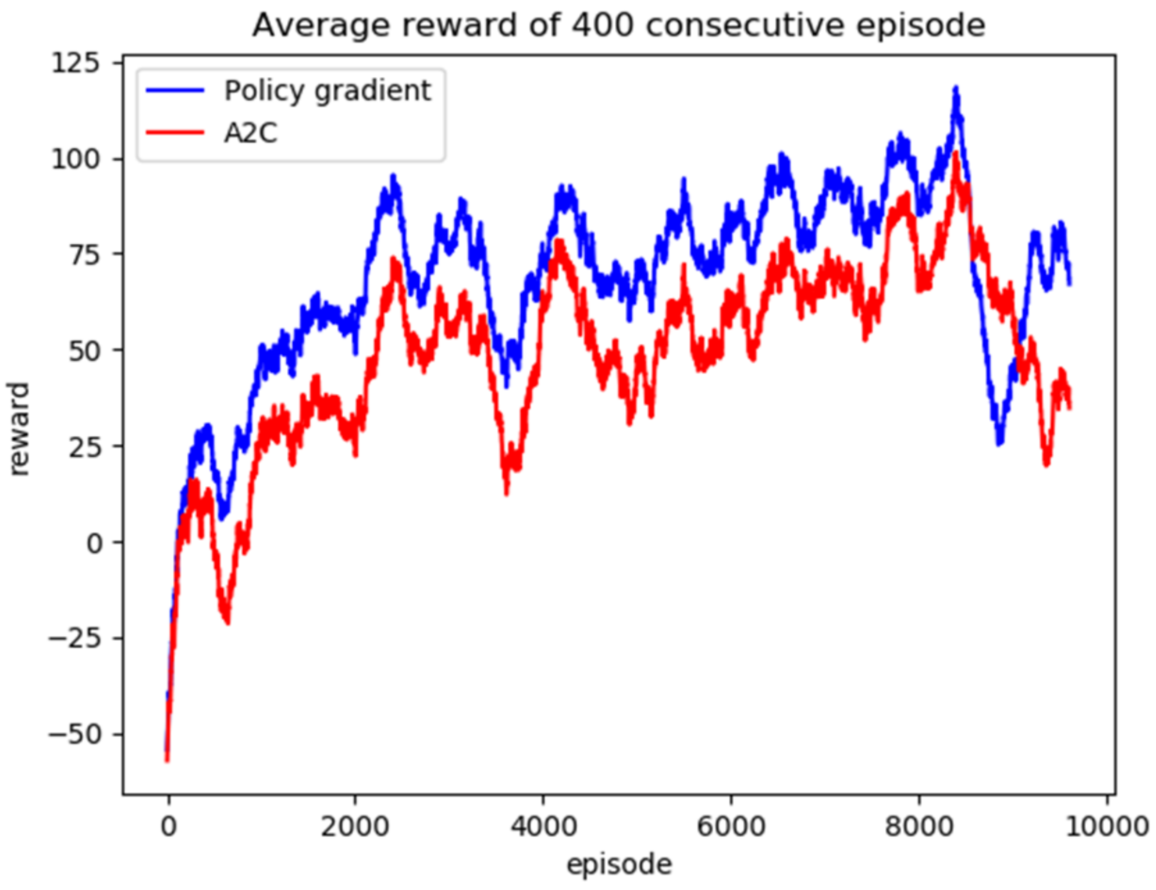


Figure 4.2:Training process of RL algorithms

Both algorithms are relatively comparable in term of convergent with policy gradient demonstrate a more aggressive fitting and A2C provide a little bit more stability.

Next we need to evaluate with the currently in use algorithm: BOLA and Throughput. To evaluate will use a cumulative distribution function due to the randomness in the network time window used for evaluation it is necessary to asset the result in a wider range of data, therefore a CDF is needed to have a correct evaluation. All algorithms is evaluate using a 1000 episode of data with random starting point in the dataset

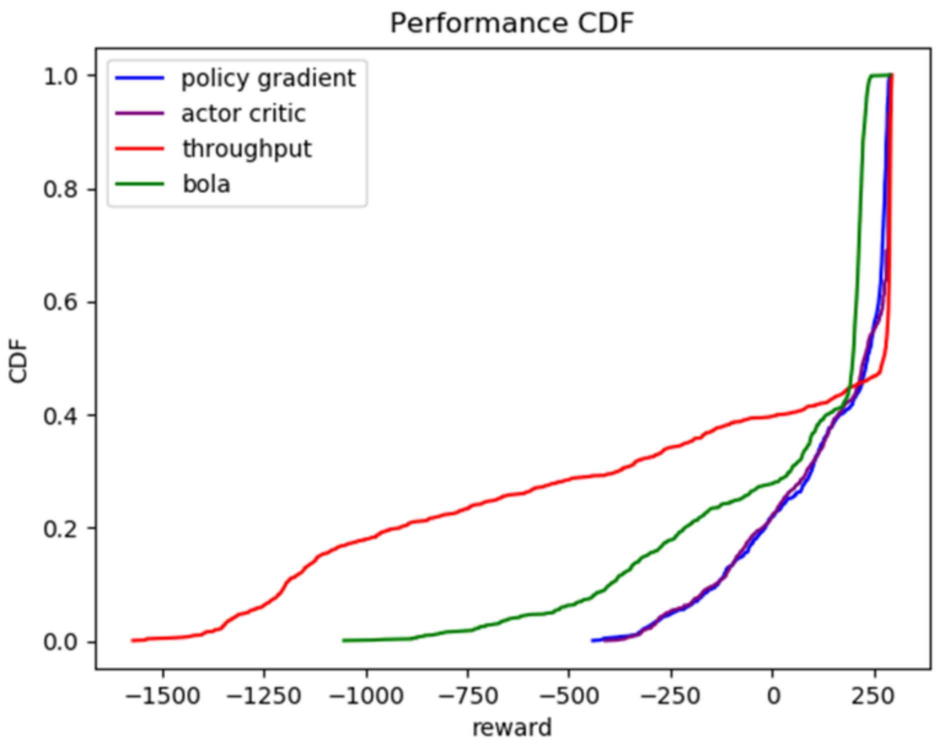


Figure 4.3: Performance evaluation

It is obvious that our reinforcement learning algorithm perform noticeably better that the other static method in both low and high bandwidth situations. The reason actor critic performance is worse in comparison with vanilla gradient decent is the fact that actor critic prioritize stability more that convergent. This attribute of actor critic can be control by the changed in the learning rate of critic as critic may allow the algorithm to converge more aggressive if it prediction is smaller. With a larger critic learning rate the problem become little bit more obvious

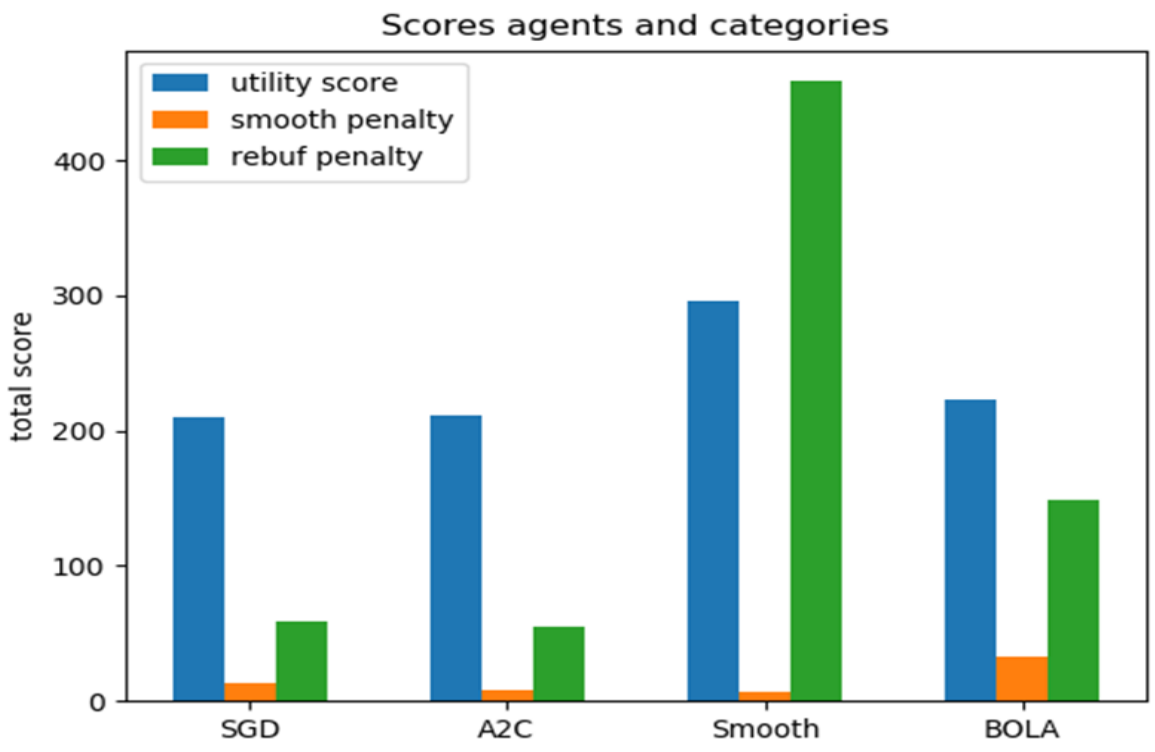


Figure 4.4: Detail analysis of each agent score

The graph above is each agent score separated into different component. In this graph the nature of individual algorithm is reveal. We can easily see that smooth throughput agent go for a very aggressive strategy which offer high utility score but also result in a huge amount of rebuff penalty. BOLA, on the other hand, uses a much less risky tactics that allow a moderate amount of utility score in exchange of a smaller rebuff penalty compare to throughput. However, it doing so with the cost of continuously switch between bitrate which make the user experience not as smooth as expected. Lastly, our two reinforcement learning agents are on the left hand show a totally different story. Despite of having the lowest utility score overall, rebuff penalty as well as switching penalty is also extremely low result in archiving the requirements that suggested through reward function (to offer maximum user experience). It is plausible to consider our agents have found a close optimal trades off between the attributes presented

Chapter 5
CONCLUSION AND FUTURE WORK

It is expected that more and more people going to have access to various mobile device in the incoming years. The extension of public wifi hotspots also contribute to the wide range and complexity of network condition an ABR have to deal with. Due to the reason above it is crucial that an upgrade for the old static ABR need to be done

This thesis have looked into one way of solving the problem using an improve ABR generated by an artificial intelligent system to generate based on specific network condition. Thesis manage to create a virtual environment that speed up the training process substantially as well as looking into two potential algorithm from reinforcement learning that being used widely as well as some of their weakness. The implement of the reward system also carefully examine in order for the algorithm reach a good performance

However, there are more that can be looked into as future works. For one thing, this thesis hasn’t produced an implementation of obtained ABR in dash.js. A future work on this issue is an interesting next step to research the effect of reinforcement learning on the ABR. Second of all, many network conditions have been simplified and most of the problem addressed in this thesis coming from the client size, a look into the situation at the server size would also give an appealing perspective in this issue.

Of course there are also numerous issues in the approach using RL itself. RL is notably a very data inefficient way to learn from a dataset, which can make it less appealing to a fully automatic training system online that continuously feed on new data to adapt to new network situation. Also there are still quite some fine tunings that needs to be done in order for the algorithm to work (exploration and exploitation trade of, state normalization, reward, ...) which can be difficult to do sometimes. Finally, new RL algorithms are being research every day and propose a unique advantage. Therefor it is natural that future works should be looking at these more advance technique to further enhance the final ABR

Works Cited

Cisco. (2020). *Cisco Annual Internet Report (2018–2023) White Paper.*

Claeys, M., Latr´e, S., Famaey, J., & Turck, F. D. (2014). Design and Evaluation of a. *IEEE COMMUNICATIONS LETTERS, VOL. 18, NO. 4*, 716-719.

D. Raca, J. Q. (2018). Beyond Throughput: a 4G LTE Dataset with Channel and Context Metrics. *In Proceedings of ACM Multimedia Systems Conference (MMSys 2018).*

Konda, V. R., & Tsitsiklis, J. N. (2000). Actor-Critic Algorithms .

Mao, H., Netravali, R., & Alizadeh, M. (2017). Neural Adaptive Video Streaming with Pensieve.

Salimans, T., & Kingma, D. P. (n.d.). Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks.

Sandvine. (2019). *Global internet phenomina report.* California.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms.

Sieber, C., Hagn, K., Kellerer, W., Moldovan, C., & Hoßfeld, T. (2018). Towards Machine Learning-Based Optimal HAS.

Spiteri, K., Sitaraman, R., & Sparacio, D. (2018). From Theory to Practice: Improving Bitrate Adaptation in the. *9th ACM Multimedia Systems Conference*.

Spiteri, K., Urgaonkar, R., & Sitarama, R. K. (2016). BOLA: Near-Optimal Bitrate Adaptation for Online Videos.

Appendix