Bitrate Adaptation for Video Streaming using Reinforcement Learning

Ayush Bhardwaj (abhardw7)*, Abdul Manan (amanan)*

CS-1470 Deep Learning

1. Introduction

We will re-implement a system from an existing paper called pensieve with pytorch. Pensieve is a system that generates adaptive bitrate algorithms using reinforcement learning. It trains a neural network model that selects bitrates for future video chunks based on observations collected by client video players. Currently, most video players use policy based algorithms to decide the bitrate for the next chunks, however, with the ever changing network conditions and with the development of new network protocols like QUIC, these policies become obsolete/ outdated. So, there is a need for an algorithm that can capture not only the diverse network protocols and conditions into its decision making process, but can also update its policies automatically over time. To address these issues, Pensieve generates ABR algorithms using observations of the resulting performance of past decisions across a large number of video streaming experiments. This allows pensieve to optimize its policy for different network characteristics and QoE metrics directly from experience.

*** Both Team Members had Equal Contribution**

2. Methodology

2.1 Pensieve Architecture

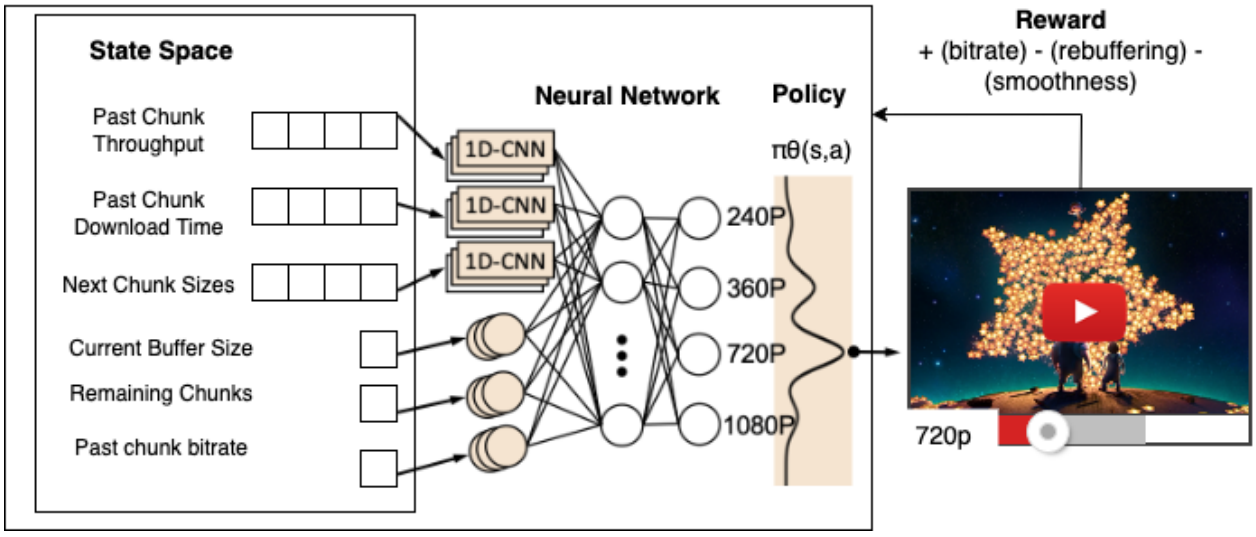

Figure: Pensieve Architecture.

Pensieve uses a deep RL algorithm to select optimal policies i.e., the bit rate to choose for maximizing the reward. In any RL system, we have two critical components (a) agent, and (b) environment. The agent interacts with an environment and at the end of each epoch or episode, the agent observes a state from the environment and takes action. As feedback, the environment returns the agent a reward signal, indicating how good the action is. The goal of the learning agent is to maximize the total reward in the environment. The state space for our agents include 6 key information: (a) Past Chunk Throughput: this is a sequence of throughput measurements,(b) Past Chunk download time: the time interval of these samples, (c) Next Chunk Sizes: Actual file size for different bitrates of the next chunk. (d) Current Buffer Size: The current playback buffer occupancy, (e) Remaining Chunks: The number chunks until the end of the video, (f) the current bitrate: to control the smoothness of the video. The reward for the RL algorithm consists of positive reward for better bitrate, penalizing rebuffering events and smoothness of the video.

*** Both Team Members had Equal Contribution**

Pensieve uses A3C (actor-critic algorithm) i.e., policy gradient method. In the Pensieve implementation it passes the past throughput measurements to a 1D convolution layer (CNN) with 128 filters (size=4, stride=1). Subsequently the chunk sizes are passed to another 1D-CNN with the same shape as stated above. Results from the previous layers are then combined with all other inputs in a hidden layer that uses 128 neurons to apply the softmax function.

2.2 **Data**

There are two primary datasets being used (a) dataset provided by FCC, and, (b) 3G/HSDPA mobile dataset collected in Norway. The first dataset from FCC has over one million throughput traces. Each of the traces logs the average throughput over 2100 seconds, at a 5 second granularity. We then extract a subset of the dataset of 1000 traces with a duration of 320 seconds. The second data set i.e. HSDPA contains 30 minutes of throughput measurements, generated using mobile devices that were streaming video while in transit. Moreover, the authors also generated network traces randomly as well using the Markovian process. We used a wider range of traces i.e. network traces representing 2G, 3G, 4G, 5G etc. These traces are usually available open-source and need preprocessing to use them in training. We trained our model for different past chunk sizes to be considered.

3. **Results**

In this section we talk about our results from our implementation. We have 4 agents and The learning rates for the actor and critic are configured to be 10^-4 and 10^–3, respectively.

*** Both Team Members had Equal Contribution**

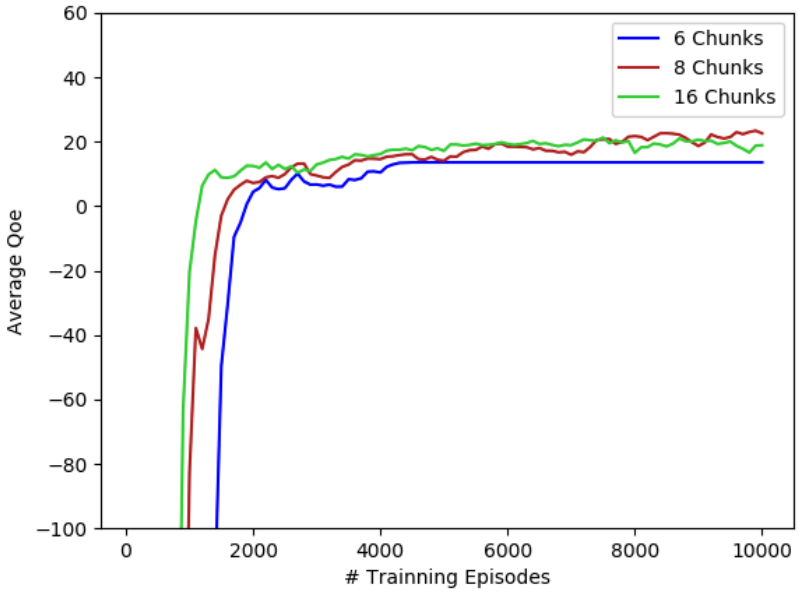

Figure: Average QoE per 100 Episodes with three throughput chunk sizes i.e., 6, 8, 16.

Past Chunk Size	Training Time
6	40 minutes
8	1 hour 12 minutes
16	10 hour 56 minutes

Table: Training Time with three throughput chunk sizes i.e., 6, 8, 16.

To have a clearer idea on how important the number of past chunks in the state space is, we ran our implementation with 3 different chunk sizes i.e., 6, 8, 16. This experiment helps us to understand how vital throughput history is in the RL loop. As shown in the above figure, considering only 6 past chunks lack appropriate information to succinctly determine network characteristics and which leads the model to eventually take a large number of episodes to converge to a decent QoE (>0). In the case of running the experiment with 8 chunks, pensieve can now extract more information using longer history and drastically improve its policy faster and converge in a smaller number of episodes but this comes at a cost of increase in training time as per the above table by 1.8x. In the last case when we provide an

*** Both Team Members had Equal Contribution**

even longer history of 16 chunks, pensieve only marginally improves its convergence and policies as compared to 8 chunks but at a very high cost in training time which is 9x the training time of the 8 chunks case.

4. **Challenges**

We had several challenges regarding pre-processing, hyperparameter choice, setting up the environment for streaming, compatibility issues for python, pytorch and understanding different variables in the functions (without any comments.) Below are a few examples about the same:

(a) First we had issues regarding converting the given traces raw data to the format of timestamp(s), throughput(Mbit/s) but after investigating we realized that every trace folder has a sample code to convert raw trace to mahimahi format which is the required format. We can calculate the throughput in a predefined window which can be used to generate the simulation format.

(b) Also we had trouble tuning the correct value for k i.e., number of past bandwidth measurements to a 1D convolution layer (CNN) but we observed that for k less than 8 there is a performance degradation, for k greater than 8 the performance remains the same. So we decided to fix this value to the minimum i.e., 8.

(c) In the initial code base there were two variables i.e., CHUNK_TIL_VIDEO_END_CAP and TOTAL_VIDEO_CHUNK with the same value and no stated difference. Then we found that CHUNK_TIL_VIDEO_END_CAP is for the learning agent and TOTAL_VIDEO_CHUNK is for the environment. They should be the same values and should be equal to the total number of chunks in the video.

(d) While setting up the environment we had issues in running Mahi Mahi as it is critical to replay those traces but it seems that mahimahi was blocking all our traffic so we had to fallback on UBUNTU 18.04 and had to do a fresh install with the entire python environment.

*** Both Team Members had Equal Contribution**

(e) Also we had compatibility issues with pytorch and some standard python2.7 libraries such as urllib3 for python 3.7.

5. **Reflection**

As we discussed in the above section we had a variety of issues pertaining to different aspects. Making Mahimahi work required a considerable amount of debugging and figuring out where things were going wrong as all our packets were dropped. Being equipped with prior system experience helped us to navigate this problem well but still it took a lot of our time. The compatibility issues that arose due to syntax translation from tensorflow to pytorch is quite significant as the type of inputs and format of outputs changed drastically for some functions. We think that the project turned out well and we were able to run some preliminary experiments for some hyperparameters as well as test our model with the actual stream. The model works well for some cases that have network conditions similar to the train data but sometimes also fails leading to high buffering. It's definitely a step in a good direction. I think if we had more time we could have run it for a largely different set of hyperparameters and traces to see how the approach works for very different scenarios.

Working through the entire DL pipeline of unpacking, preprocessing and writing the model was an enriching experience. A cherry on the top was that we can try our model with actual video streams as that part of the process comes with its own challenges. We learned about several technologies like mahimahi, networking apart from building over our core DL principles. Overall we enjoyed every bit of the process and gained a substantial amount of knowledge from this project.

*** Both Team Members had Equal Contribution**

6. **Ethical Implications**

The author uses two primary datasets (a) dataset provided by FCC, and, (b) 3G/HSDPA mobile dataset collected in Norway. The first dataset from FCC has over one million throughput traces. Each of the traces logs the average throughput over 2100 seconds, at a 5 second granularity. The author's then extracted a subset of the dataset of 1000 traces with a duration of 320 seconds, from the August 2016 collection. The second data set i.e., HSDPA contains 30 minutes of throughput measurements, generated using mobile devices that were streaming video while in transit. The method of collection seems appropriate but we don't think it's representative as there is a strong digital divide between developing nations and developed nations. For example, the infrastructure of developing nations like pakistan, South Africa, India still use a lot of copper based connection for last-mile delivery which puts a strong upper bound on throughput so the network characteristics are completely different. This raises an important question: is the architecture or project biased towards users of developed countries as opposed to developing countries with lower throughputs, low-end devices and older generation of mobile networks.

The stakeholders are end-user clients that use different video streaming applications on any device. This could range from casual cases e.g, a user watching youtube to critical users e.g., doctors performing remote surgeries with different experts. In the latter use case if pensive doesn't have good decision making the surgery might be buffered or could have poor video quality which could impact the quality of surgery or even impose threat on the life of the patient.

*** Both Team Members had Equal Contribution**

7. Related Work

There are other existing works that apply RL to adaptive video streaming [1, 2, 3, 4]. However, all of these schemes apply RL in a tabular form, which stores and learns the values for all states and actions explicitly, rather than using NN. As a result, these schemes do not scale to the large state spaces necessary for good performance in real networks, and their evaluation has been limited to simulations with synthetic network models. For example, the most recent tabular scheme [1] relies on the fundamental assumption that network bandwidth is Markovian, i.e., the future bandwidth depends only on the throughput observed in the last chunk download. This assumption confines the state space to consider only one past bandwidth measurement, making the tabular approach feasible to implement. However, previous research has shown that the information contained in one past chunk is not sufficient to accurately infer the distribution of future bandwidth.

1. Federico Chiariotti, Stefano D'Aronco, Laura Toni, and Pascal Frossard. 2016. Online learning adaptation strategy for DASH clients. In Proceedings of the 7th International Conference on Multimedia Systems (MMSys '16). Association for Computing Machinery, New York, NY, USA, Article 8, 1–12.
2. Claeys, M., Latré, S., Famaey, J., Wu, T., Leekwijck, W.V., & Turck, F.D. (2013). Design of a Q-learning-based client quality selection algorithm for HTTP adaptive video streaming. ALA 2013.
3. Maxim Claeys, Steven Latré, Jeroen Famaey, Tingyao Wu, Werner Van Leekwijck, and Filip De Turck. 2014. Design and optimisation of a FAQ-learning-based HTTP adaptive streaming client. Connect. Sci 26, 1 (March 2014), 25–43.
4. J. van der Hooft, S. Petrangeli, M. Claeys, J. Famaey and F. De Turck, "A learning-based algorithm for improved bandwidth-awareness of adaptive streaming clients," 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), 2015, pp. 131-138

*** Both Team Members had Equal Contribution**

8. **Existing Implementations**

- https://github.com/hongzimao/pensieve

*** Both Team Members had Equal Contribution**