PnP-DRL: a Plug-and-Play Deep Reinforcement Learning approach for Experience-Driven Networking

Zhiyuan Xu, Kun Wu, Weiyi Zhang, Jian Tang, Yanzhi Wang, Guoliang Xue

***Abstract*—While Deep Reinforcement Learning has emerged as a *de facto* approach to many complex experience-driven networking problems, it remains challenging to deploy DRL into real systems. Due to the random exploration or half-trained deep neural networks during the online training process, the DRL agent may make unexpected decisions, which may lead to system performance degradation or even system crash. In this paper, we propose *PnP-DRL*, an *offline-trained, plug and play* DRL solution, to leverage the batch reinforcement learning approach to learn the best control policy from pre-collected transition samples without interacting with the system. After being trained without interaction with systems, our Plug and Play DRL agent will start working seamlessly, without additional exploration or possible disruption of the running systems. We implement and evaluate our PnP-DRL solution on a prevalent experience-driven networking problem, Dynamic Adaptive Streaming over HTTP (DASH). Extensive experimental results manifest that 1) The existing batch reinforcement learning method has its limits; 2) Our approach PnP-DRL significantly outperforms classical adaptive bitrate algorithms in average user Quality of Experience (QoE); 3) PnP-DRL, unlike the state-of-the-art online DRL methods, can be off and running without learning gaps, while achieving comparable performances.**

***Index Terms*—Experience-driven Networking, Deep Reinforcement Learning, Batch Reinforcement Learning.**

I. Introduction

Recent years have witnessed dramatic increases in the complexity and dynamics of modern communication networks. Due to the essence of highly dynamic time-variant of communication networks, traditional pre-defined and fixed control policies [1], [2], [3] may lead to poor performances because of failing to adapt to run-time states of the network environment. Optimization-based control policies [4], [5], [6], which have been proposed to consider system dynamics, rely on accurate mathematical environment model or pre-assumptions about the environment. The recent breakthrough of Deep Reinforcement Learning (DRL) enables model-free experience-driven networking to train control policies that outperform human-engineered heuristics for a wide range of tasks in computer systems and networking, such as routing and traffic engineering [7], [8], [9], video streaming [10], [11]. Compared with other learning-based methods [12], [13], [14], such as supervised learning, which are more concerned with predictions on some key factors, DRL is more suitable to deal with sequential decision making tasks with the objective of optimizing long-term performance in the face of system dynamics and changes along with each decision. In DRL, the agent learns the best policy to control the system or allocate resources through interacting with the environment in a trial-and-error manner. Practical deployment of these RL policies, however, faces justified skepticism about their robustness when they face unusual situations. One significant barrier to applying online methods on real-world problems is the difficulty of large-scale online data collection. Safety with online RL is another critical issue as RL fundamentally must explore and make mistakes in order to learn from them. While these necessary mistakes are not problem in an offline simulator, they are dangerous to a running system, and may cause unexpected consequences which disrupt a functional system. For example, it would be unreasonable to allow RL agents to explore, make mistakes, and learn while controlling an autonomous vehicle or treating patients in a hospital. Even for non-life-threatening applications, like video streaming, online DRL could still be unacceptable. According to a survey [15], the slow startup of a video, due to data collection and online training, will cause user abandonment. The authors observed that "*After 10 seconds of startup delay, more than half of your audience has left, and only 8% of users will return to your website within 24 hours after experiencing a video failure*". Besides, if the video experience were not satisfying, most users will close the video [16], among them only 49% would try again immediately, meaning that another half of the audience will be lost. Even more, 29% of the audience would seek different service providers, so the website will loose nearly one third to competitors.

This makes learning from pre-collected experience a promising idea. Batch Reinforcement Learning (batch RL) was proposed to learn from a fixed dataset without further interactions with the environment. To utilize existing large datasets for networking applications, the ability for batch RL algorithms to learn offline from these datasets has an enormous potential impact in shaping the way we build machine learning systems for the future. However, Batch RL has its own limits when applying to networking problems: (1) large data are commonly cheaply collected and sub-optimal; (2) in real-world datasets, it is common that data distributions are

Zhiyuan Xu, Kun Wu, and Jian Tang are with the Department of Computer Science and Engineering, Syracuse University, USA. Email: {zxu105, kwu102, jtang02}@syr.edu. Weiyi Zhang is with the Network Evolution Strategies, LLC, Holmdel, NJ, USA. Email: zhzing@pm.me. Yanzhi Wang is with the Department of Electrical and Computer Engineering, Northeastern University, USA. Email: yanz.wang@northeastern.edu. Guoliang Xue is with the School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, USA. Email: xue@asu.edu.

This research was supported in part by NSF grants 1704662 and 1704092. The information reported here does not reflect the position of the government.

narrow and more often, biased, i.e., only cover a (limited) subset of state and action space of DRL; (3) some data are not representable by policies, such as human demonstrations, hence, cannot be observed by offline agents. These limits will lead to *extrapolation error [17]* that prevents the DRL agent from adapting to the environment when plugged to online systems.

To address such limitations, many batch RL algorithms have been proposed. For example, Fujimoto [17] recently introduced Batch-Constrained deep Q-learning (BCQ), which aims to minimize the mismatch between the state-action visitation of the current policy and the state-action pairs contained in the dataset. Although these algorithms were demonstrated well in video gaming applications [17], [18], our investigation found out that a straightforward application of it to networking problem does not work. The reason for the unsuccessfulness of these batch RL algorithms in the networking domain is twofold: In theory, a simple fully-connected or convolutional neural network (e.g., used in BCQ [17]) often fails to capture the hidden features from the high-dimensional time-series state of networking systems; In practice, it is highly likely that the numbers of collected transition samples for various control actions are severely imbalanced, thus may cause serious bias to the DRL agent during the training. In this paper, we propose *PnP-DRL*, an *offline-trained, plug and play* DRL solution, which consists of two new techniques to solve the aforementioned two issues: 1) a Long-Short-Term-Memory (LSTM) based state representation network for better feature extraction ability; and 2) a novel weighted loss function to train a generative model $G(\cdot)$, adapting the BCQ approach to practical networking systems. Intuitively, LSTM-based representation network is capable of handling temporal data in state, i.e., analyzing the relationship between time-series data, which provides more powerful learning ability to the DRL agent. Meanwhile, a weighted loss function could help to mitigate the issue of biased/narrow data distributions by encouraging to learn more from less seen actions, thus improving the generalization ability of the DRL agent. Combining these two mechanisms is able to further potentially improve the performance of current batch RL algorithm in networking control tasks.

To the best of our knowledge, this is the first work to apply and refine the emerging batch RL algorithm to address experience-driven networking problems. We summarize our contributions as follows:

- We demonstrate that online DRL solution could be unstable and even dangerous to functioning systems.
- We propose to use offline training for a *plug and play DRL agent*, which can be applied into networking system seamlessly and take effect promptly without further interactions with environment.
- We present two new techniques, *LSTM-based representation network* and *transitions imbalanced training*, to improve BCQ so that a plug and play DRL agent can be enabled.
- Extensive experiments demonstrate that our plug and play DRL agent (1) behaves consistently without any disruption of the system (2) outperforms baseline ABR algorithms while achieving comparable performance to the online DRL solutions.

The rest of the paper is organized as follows. We introduce the necessary background about DRL and batch reinforcement learning in Section II. The overview and design of PnP-DRL is presented in Section III. We introduce a case study in Section IV, and present experimental results and corresponding analysis in Section V. We review related works and conclude the paper in Section VI and VII, respectively.

II. Background and Challenges

A. Online Deep Reinforcement Learning

In a common online DRL setting, a DRL agent needs continuous interaction with an environment in discrete decision epochs. At decision epoch t, the agent observes the state of the environment $\mathbf{s}_t \in \mathcal{S}$ and makes an action decision $\mathbf{a}_t \in \mathcal{A}$ based on the observed state and its control policy $\pi(\cdot) : \mathcal{S} \rightarrow \mathcal{A}$. After executing the action on the environment, the DRL agent receives an immediate reward $r_t \in \mathbb{R}$, and the environment transfers to a new state $\mathbf{s}_{t+1}$. The goal of DRL is to find a policy $\pi(\cdot)$ that can maximize the sum of discounted rewards $R_t = \sum_{i=t+1}^{\infty} \gamma^i r_i$, where $\gamma \in (0, 1]$ is a discount factor determining how much the agent cares about future rewards. A pair of $\{\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1}\}$ is also called a transition sample.

Mnih *et al.* [19] proposed Deep Q-Network (DQN) to estimate the expected discounted cumulative reward (called *Q value*), which extended Deep Neural Network (DNN) into traditional tabular-based Q-learning as the function approximator. More specifically, given a pair of state and action $(\mathbf{s}_t, \mathbf{a}_t)$, DQN can derive the corresponding Q value:

$$Q(\mathbf{s}_t, \mathbf{a}_t; \boldsymbol{\theta}) = \mathbb{E}\Big[R_t | \mathbf{s}_t, \mathbf{a}_t\Big], \tag{1}$$

where $\boldsymbol{\theta}$ represent the weights of DQN. The optimal control policy $\pi(\cdot)$ is then defined as:

$$\pi(\mathbf{s}_t) = \text{argmax}_{\mathbf{a} \in \mathcal{A}} Q(\mathbf{s}_t, \mathbf{a}; \boldsymbol{\theta}). \tag{2}$$

The weights $\boldsymbol{\theta}$ of DQN $Q(\cdot)$ are updated by minimizing the mean square error (MSE) loss function:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}\big[(r_t + \gamma Q(\mathbf{s}_{t+1}, \pi(\mathbf{s}_{t+1}); \boldsymbol{\theta}) - Q(\mathbf{s}_t, \pi(\mathbf{s}_t); \boldsymbol{\theta}))^2\big]. \tag{3}$$

Moreover, Mnih *et al.* [19] further presented the experience replay buffer and target networks to improve the learning stability of DQN.

Challenges: In reality, most of the DRL methods require training on enough transition samples before getting ready to use (i.e., finding a good control policy). Hence it typically takes a long time for a control decision to take effect and collect feedbacks to the DRL agent [20]. This is not endurable, especially in real commercial systems. Because it is impossible to expect real systems to halt and wait for a reasonable solution. Moreover, during the online training, it is unavoidable for DRL agents to make random decisions because of 1) neural networks used by DRL agents are usually required to be initialized randomly at the beginning of trainings; 2) DRL agents will try random explorations, i.e., trying different

potential actions under various states to check whether these actions can lead to high/low rewards, such as ϵ-greedy in DQN [19] and Ornstein-Uhlenbeck (OU) process in Deep Deterministic Policy Gradient (DDPG) [21]. However, in most of the practical systems, these random decisions can be costly or possibly unsafe. A functioning system can be impacted and may even get crashed due to unexpected actions.

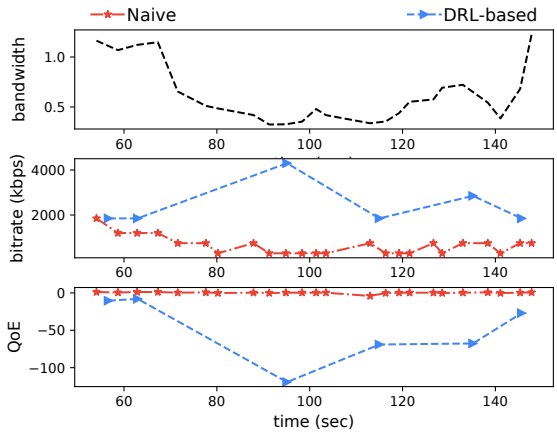


Fig. 1: System disrupted during online training process

We use Fig. 1 to demonstrate our observations with an example of Dynamic Adaptive Streaming over HTTP (DASH) [11]. When network bandwidth starts dropping from 60s to 100s (as shown in the first row), it indicates the network condition is getting worse. To ensure the user's QoE, the rule-based baseline Adaptive BitRate (ABR) algorithm [6] (i.e., in red line), naturally, keeps selecting a low bitrate for each video chunk. However, due to the *random exploration*, the DRL agent (i.e., in red line), counterintuitively, chooses a large bitrate regardless of the current network condition (as shown in the second row). Consequently, we can observe a significant performance degradation with DRL-based algorithm during this time period (as shown in the third row).

Therefore, it remains a challenge to train a DRL agent to achieve state-of-the-art performance while avoid disrupting the system during training.

B. Batch Reinforcement Learning

Batch reinforcement learning [22] was proposed for scaling reinforcement learning to tasks where the online training procedure is costly, risky, or time-consuming. In batch reinforcement learning, the agent tries to learn control policy directly from *pre-collected datasets (trajectories) without further interactions with the environment*. However, as shown in [23], a directly supervised behavioral cloning from pre-collected datasets (sub-optimal or even expert-level trajectories) could not lead to satisfying performance. One of the main causes is due to the aforementioned *extrapolation error*, a phenomenon in which unseen state-action pairs are erroneously estimated to have unrealistic values.

To mitigate the above issue and enable learning from sub-optimal trajectories in Batch RL, Fujimoto *et al.*[17] proposed Batch-Constrained deep Q-learning (BCQ). In BCQ, the DRL agent learns to maximize reward while minimizing the mismatch between the state-action derived by its control policy and the state-action pairs contained in trajectories. More specifically, BCQ proposes a state-conditioned Generative Network (GN) $G(\mathbf{a}_t|\mathbf{s}_t;\boldsymbol{\xi})$ with weights $\boldsymbol{\xi}$ to calculate the probability of each action under a given state $\mathbf{s}_t$. Then BCQ uses a threshold δ to filter out all potential actions for the state. To adapt and adjust the threshold, the output probabilities of each action are scaled by the maximum probability from the GN over all actions.

$$A_t = \mathbf{a}|G(\mathbf{a}|\mathbf{s}_t;\boldsymbol{\xi})/\max_{\hat{\mathbf{a}}\in\mathcal{A}} G(\hat{\mathbf{a}}|\mathbf{s}_t;\boldsymbol{\xi}) > \delta, \tag{4}$$

where δ is the selection threshold indicating how much we trust these actions, $\delta = 0$ equates to the normal Q-learning and $\delta = 1$ represents the normal imitation learning. $\mathcal{A}$ is the whole action space. The optimal control policy $\pi(\cdot)$ is then modified to:

$$\pi(\mathbf{s}_t) = \text{argmax}_{\hat{\mathbf{a}}\in A_t} Q(\mathbf{s}_t, \hat{\mathbf{a}}; \boldsymbol{\theta}), \tag{5}$$

while DQN is updated by the same way as Eq. (3), the GN is updated in a standard supervised manner, the actions in trajectories will be used as ground-truth labels. More specifically, the GN is updated by minimizing the negative log-likelihood (NLL) loss function:

$$\mathcal{L}(\boldsymbol{\xi}) = \mathbb{E}\big[-\log G(\mathbf{a}_t|\mathbf{s}_t;\boldsymbol{\xi})\big]. \tag{6}$$

Compared with the vanilla DQN that iterates the whole action space to select the action with the maximal Q value, BCQ reduces the searching space to the state-related actions and eliminates the actions which are unlikely to be selected by the control policy $\pi(\cdot)$, thus improving the learning efficiency.

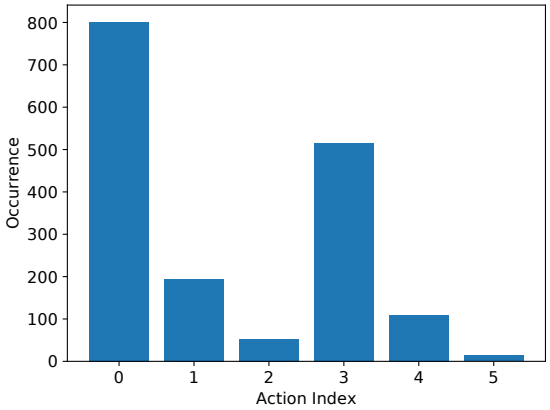


Fig. 2: Transition imbalance

Challenges: However, BCQ is not without issue, when it comes to networking problems. Based on our investigation, there are two major issues which cause BCQ cannot be applied for experience-driven networking.

- **Network Representation:** In BCQ [17], for robotic control applications, the authors just applied a two layer fully-connected neural networks as the function approximator, which is sufficient for their data analysis. However, in a practical networking environment, the state

of a system is relatively high-dimensional and time-dependent. BCQ is not effective to capture the state of networking systems with time-variant data. For better network representation, a more powerful neural network is needed.

- **Transition Imbalance:** In BCQ [17], the application studied happened to have evenly distributed transitions. However, in practical network systems, transitions collected from an existing control policy may have imbalanced distributions. For example, in an un-congested network, a sender may always choose a maximal sending rate to send data. Thus, in the transitions collected from such an environment, we may hardly see a low sending rate action (and its corresponding state). In this case, for BCQ, trained with imbalanced dataset, its DRL agent may fail to adapt to the system dynamic.

To demonstrate the transition imbalance phenomenon, we ran a baseline ABR algorithm [6] for DASH, based on a commonly-used trace *Norway 3G dataset* [24]. Depending on the network status, the collected transitions most likely may not be uniformly distributed. Fig 2 shows the numbers of collect control transitions from DASH's six actions. As we can see from the uneven pattern of the collected transitions, these transitions are not equally sampled and imbalanced, which could spoil the training result.

III. Architecture and Design of PnP-DRL

In this paper, we propose to leverage a state-of-the-art batch reinforcement learning method, *Batch-Constrained Q-learning* (BCQ), to train a DRL agent from pre-collected historical decision transitions without any further interactions in the system. In this way, we provide a *plug and play agent*, which, after offline training, can be plugged into systems and take effect immediately, *without learning gap and performance gaffes*. Plug and play agent is a general solution for DRL offline training, independent of system, platform and problems. Fig. 3 illustrates the architecture of our solution, PnP-DRL, which consists of three key components:

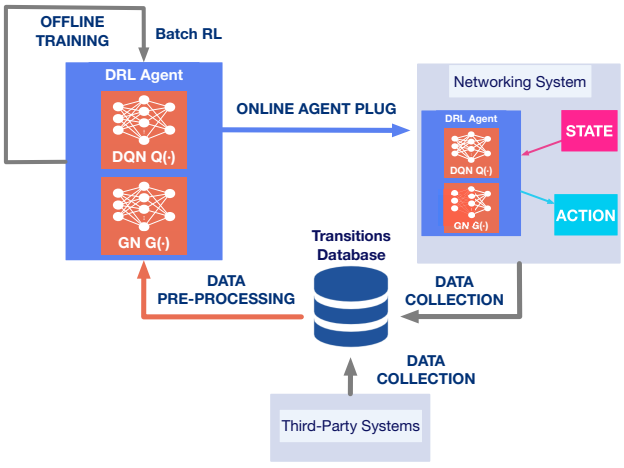


Fig. 3: The architecture of PnP-DRL

- **Data Collection:** Data collection is a continuous operation which collects all related information, based on existing control policies, from a running networking system. It worth noting that this data collection component is independent to specific network problems. Based on the policy setup, it collects all the information which is general for various applications. For example, for a networking system, this component can continuously collect *size of congestion window, estimated network bandwidth, data arrival rate, user's QoE, signal-to-noise ratio, buffer size etc*. Also, the data is not necessarily collected from the current system (that we want to optimize). Instead, the data can come from any existing third-party system that provides useful information. When a learning problem is identified, PnP-DRL will retrieve related data from the transitions database, process and sanitize it, and then feed into the offline training component.
- **Offline Training:** Without interacting and potentially disrupting a networking system, this component only analyzes a dataset of transition samples $\{\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}'_t\}$, which are collected from existing control policies. Each transition sample records that after executing the action $\mathbf{a}_t$ at a discrete decision epoch t, the state of the environment $\boldsymbol{s}_t$ transfers to a new state $\boldsymbol{s}_{t+1}$, and the DRL agent receives a reward r_t. The DRL agent offline learns the best networking control policy from these transition samples. At this step, no further interactions with the system, nor new generated transition samples are needed.
- **Online Agent Plug:** After training, we can directly plug the DRL agent into the system, and expect it to achieve state-of-the-art performance and adapt to the dynamics of the system.

The architecture of PnP-DRL demonstrates the processing steps to deploy a plug and play agent: 1) collect statistical data from the current system or third-party systems; 2) pre-process the transition samples according to the definition of state, action, and reward of the DRL agent; 3) train the DRL agent offline with those collected transition samples; 4) plug the DRL agent into a running networking system and make online decisions without interruption.

IV. DASH: A Case Study

To demonstrate how the proposed plug and play DRL approach works, we use Dynamic Adaptive Streaming over HTTP (DASH) as a case study. DASH is one of the most dominant solutions for video transmission nowadays [25]. The DASH server is essentially an HTTP server that hosts the media segments, such as all video chunks with all potential bitrates, and records their information and URLs into a Media Presentation Description (MPD) file. Meanwhile, the DASH server makes the MPD file and these video chunks available to its clients as HTTP resources, and clients sequentially download each video chunk from the DASH server. Extensive research efforts have been made to develop Adaptive Bit Rate (ABR) algorithms, which aim to fully utilize network resources and optimize user Quality of Experience (QoE) for the whole video. In ABR algorithms, the client takes current network status and video content information into consideration, and then decides to fetch the video chunk with the most proper quality (such as bitrate).

Recently, DRL-based ABR algorithms have been presented [10], [11]. Unlike traditional rule-based or heuristic-based solutions, DRL-based methods do not rely on accurate mathematical model of the system. Instead, online DRL solutions try to learn the best control policy via interacting with the system in a trial-and-error manner, enabling experience-driven control. Though these work showed promising final results, such practice can be risky and may not be feasible to explore in real environment. We observed that the online DRL-based ABR algorithm, which gradually learns to make better ABR decisions through reinforcement, in the form of reward signals that reflect user QoE for past decisions, could experience severe performance degradation during the training. For example, as shown in Fig. 1, an online training agent may suggest to try a long-time buffering and learn the rewards of such behavior. However, users usually will not tolerate such long buffering time and may directly close the video player instead of keep waiting. Therefore, in reality, it may be necessary to learn control policies from the offline dataset instead of directly interacting with the system.

In DASH, we first collect transition samples from existing ABR algorithms. Since these algorithms are already implemented in the system, and we just collect the data, there is no extra burden on the system. It is worth noting that in DASH, these existing DRL-based ABR algorithms do not always make *good* decisions. In some circumstances, they may achieve sub-optimal or even non-satisfactory performance. Therefore, the collected transition samples are not perfect and most of time sub-optimal. Consequently, we noticed that, shown in our experiments, the state-of-the-art batch RL does not work well for DASH, better solution is needed.

A. Proposed Plug and Play DRL Approach

We first present the definition of **state**, **action** and **reward** for the DRL agent in DASH.

State: at each decision epoch t, the state is defined as $\mathbf{s}_t = (\vec{x}_t, \vec{d}_t, \vec{n}_t, b_t, c_t, l_t)$, where $\vec{x}_t = \{x_t^1, x_t^2, ..., x_t^k\}$ and $\vec{d}_t = \{d_t^1, d_t^2, ..., d_t^k\}$ are the measured network throughput and download time for the past k video chunks, respectively; $\vec{n}_t = \{n_t^1, n_t^2, ..., n_t^m\}$ consists of m available sizes for the next video chunk; b_t is the size of current buffer; c_t is the remaining number of video chunks; and l_t is the selected bitrate for the last downloaded video chunk.

Action: the action at each decision epoch a_t indicates the choice of the bitrate for the next video chunk. Note that the available bitrates for the video are pre-defined by the video server, the DRL agent has m available choices for the next video chunk.

Reward: we use the following widely-used QoE metric as the reward, which is consistent with most existing work [11], [6].

$$r = M_t - \mu_1 T_t - \mu_2 |M_t - M_{t-1}|, \tag{7}$$

where M_t is the selected bitrate of video chunk at decision epoch t, T_t is the rebuffering time, and the last term encourages smooth transition between chunks and penalties the sudden changes of video quality. μ_1 and μ_2 are two variables to balance the relative importance of rebuffering time and quality change. The QoE metric reflects user preferences for the system. For example, in DASH, during the video playing, the user prefers a larger video bitrate, less re-buffering time, and smooth transition between video chunks, which are all considered in the defined QoE metric (reward for the DRL agent). In this paper, we focus on the offline training of a DRL agent, by following the same definition of state, action and reward in work [11] for the DASH problem.

If we collect enough transition samples $\{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\}$ from the existing control policies in a running system, we can start to offline train a DRL agent without interactions with the system. However, as explained in Secton II, a straightforward application of state-of-the-art batch reinforcement learning algorithm (i.e., BCQ) to the DASH problem can not necessarily lead to a satisfactory performance. To solve these two issues, we propose two new techniques: (1) *a LSTM-based state representation network for better feature extraction ability*; and (2) *a weighted loss function to train the generative model $G(\cdot)$ of BCQ*. It is worth noting that although these proposed techniques are designed for the DASH problem in this work, they are generic and can be easily applied to other systems with similar issues.

B. Representation Network

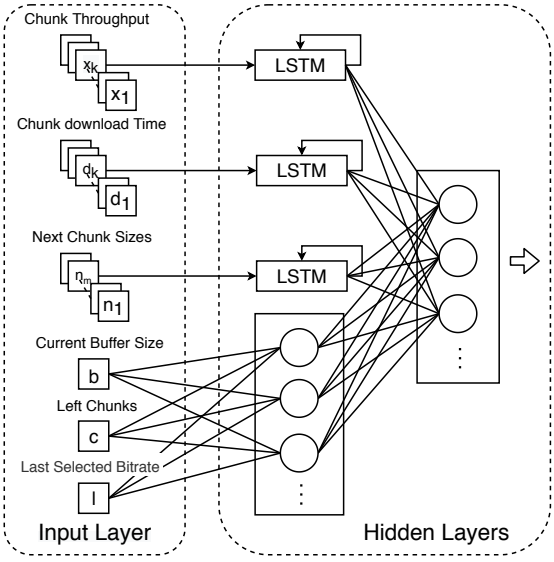


Fig. 4: LSTM-based Neural Network Structure

To have a better representation for the **state**, we propose a novel LSTM-based neural network to extract features from high-dimensional time-series state. Compared with fully-connected neural networks, LSTM is capable of learning long-term dependencies from time-series input. The state is split into two sets:

1) the time-series set $(\vec{x}_t, \vec{d}_t, \vec{n}_t)$ that includes historical or future information; and
2) the flat set (b_t, c_t, l_t) that includes other constant information.

As illustrated by Fig. 4, the first set is connected to three LSTM layers and the second set is connected to fully-connected layers, then the output of these two paths are

concatenated together as a hidden layer to further connected to a fully-connected layer to get the representation. All the LSTM layers and fully-connected layers will be trained together in an end-to-end manner.

C. Training with Transitions Imbalance

It is well known that imbalanced dataset will degrade the generalization of DNN models, making it hardly adapt to unknown circumstances [26]. Therefore, to improve the generalization ability of the DRL agent, we propose to use a weighted loss function to alleviate this issue. Instead of directly applying the NLL loss function defined in Eq. (6) to train GN, we add a weight to each training target,

$$\mathcal{L}(\boldsymbol{\xi}) = -\frac{1}{K}\sum_{i=1}^{K} w_i \cdot \log G(\mathbf{a}_i|\mathbf{s}_i;\boldsymbol{\xi}), \tag{8}$$

where the weight w_i is calculated based on collected transitions:

$$w_i = 1 - \frac{o(a_i)}{N} \tag{9}$$

where $o(a_i)$ is the number of transitions that contain the action a_i in the whole dataset, N is the total number of transitions in the dataset. Eq. (9) assigns more weights to the less sampled actions, hence encourages the DNN not to ignore, but rather to learn more from the less-sampled transitions.

For the completeness of the presentation, we formally present our *PnP-DRL* in **Algorithm** 1. After collecting transition samples from the running system and storing them into the replay buffer (Lines 1-2), the offline training procedure starts with randomly initializing DQN and GN, and copying the weights of DQN to its target network (Lines 3-4). For each training epoch and each mini-batch of sampled transitions (Line 6), we first filter out the potential actions based on their probability and a threshold with Eq. (4) (Line 7). Then the target action and its corresponding target value for training DQN can be computed by Eq. (3) (Lines 8-9). DQN then will be trained with minimizing the mean square error (MSE) loss function (Line 10), and GN is set to be trained by minimizing the weighted Negative Log-Likelihood (NLL) loss function, defined in Eq. (8), to mitigate the transition imbalance issue (Lines 11-12). The target network of DQN is slowly updated, with a small target update rate τ (Line 13) as the control parameter. After the offline training, the DRL agent will be directly plugged into a running system (Line 15), seamlessly start monitoring run-time system state, and promptly make decisions and execute actions without learning gap and further interacting with the system (Lines 17-19).

Algorithm 1: PnP-DRL

Input: The number of training epochs T, replay buffer $\mathcal{B}$, the size of a mini-batch K, action filter threshold δ, target update rate τ, DQN $Q(\cdot)$ with weights $\boldsymbol{\theta}$ and its target network $Q'(\cdot)$ with weights $\boldsymbol{\theta}'$, GN $G(\cdot)$ with weights $\boldsymbol{\xi}$.

Output: $Q(\cdot)$ and $G(\cdot)$ with weights $\boldsymbol{\theta}$ and $\boldsymbol{\xi}$.

********** Data Collection **********

1 Collect desirable information from the running system with existing control policies;

2 Pre-process and store the transition samples into the replay buffer $\mathcal{B}$;

********** Offline Training **********

3 Randomly initialize the weights $\boldsymbol{\theta}$ and $\boldsymbol{\xi}$;

4 Update the target network $\boldsymbol{\theta}' := \boldsymbol{\theta}$;

5 **for** *training epoch* $1 \rightarrow T$ **do**

6 Sample K transitions $(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)$ from $\mathcal{B}$;

7 Filter out potential target actions with threshold δ $A_i := G(\mathbf{a}'_i|\mathbf{s}'_i;\boldsymbol{\xi}) / \max_{\hat{\mathbf{a}}_i \in \mathcal{A}} G(\hat{\mathbf{a}}_i|\mathbf{s}'_i;\boldsymbol{\xi}) > \delta$;

8 Compute the target action for each transition $\mathbf{a}'_i := \text{argmax}_{\hat{\mathbf{a}}_i \in A_i} Q(\mathbf{s}'_i, \hat{\mathbf{a}}_i;\boldsymbol{\theta})$;

9 Compute the target value for each transition $y_i := r_i + \gamma Q'(\mathbf{s}'_i, \mathbf{a}'_i;\boldsymbol{\theta}')$;

10 Update weights $\boldsymbol{\theta}$ of DQN with loss function $\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{K}\sum_{i=1}^{K}(y_i - Q(\mathbf{s}_i, \mathbf{a}_i;\boldsymbol{\theta}))^2$;

11 Compute the sample imbalance weight w_i;

12 Update weights $\boldsymbol{\xi}$ of GN with loss function $\mathcal{L}(\boldsymbol{\xi}) = -\frac{1}{K}\sum_{i=1}^{K} w_i \cdot \log G(\mathbf{a}_i|\mathbf{s}_i;\boldsymbol{\xi})$;

13 Update the weights of target network $\boldsymbol{\theta}' := \tau\boldsymbol{\theta} + (1-\tau)\boldsymbol{\theta}'$;

14 **end**

********** Online Agent Plug **********

15 Plug DQN and GN to into the running system;

16 **while** *Observed state* s_t *from the system* **do**

17 Filter out potential actions with threshold δ $A_t := G(\mathbf{a}_t|\mathbf{s}_t;\boldsymbol{\xi}) / \max_{\hat{\mathbf{a}}_t \in \mathcal{A}} G(\hat{\mathbf{a}}_t|\mathbf{s}_t;\boldsymbol{\xi}) > \delta$;

18 Select the action with the highest Q value $\mathbf{a}_t := \text{argmax}_{\hat{\mathbf{a}}_t \in A_t} Q(\mathbf{s}_t, \hat{\mathbf{a}}_t;\boldsymbol{\theta})$;

19 Execute the action $\mathbf{a}_t$ in the system;

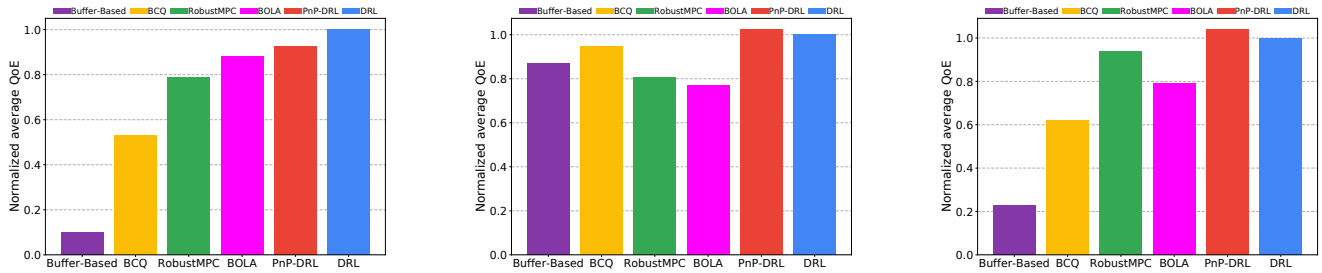
20 **end**

V. Performance Evaluation

A. Implementation

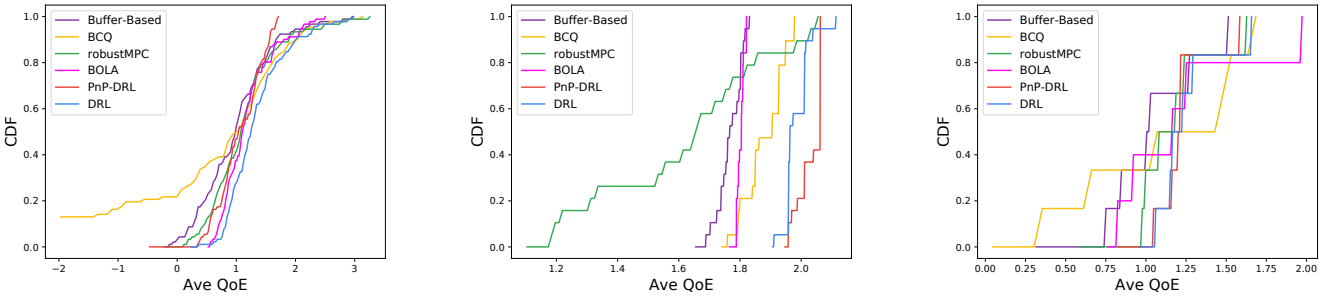
We implemented the proposed PnP-DRL using PyTorch [27]. For DQN and GN, we used 64 neurons on each LSTM layer and 64 neurons on each ReLu-activated [28] fully-connected layer. During the offline training, we used Adam [29] as the optimizer, and the learning rate is set to 0.001. Following the commonly-used settings for all experimental scenarios, the reward discount factor γ is set to 0.99, the target network updating rate τ is set to 0.005, the action filter threshold δ is set to 0.45, and the mini-batch size is set to 256. The QoE weights μ_1 and μ_2 are set to 4.3 and 1, respectively.

PnP-DRL is implemented as a standalone HTTP server (a.k.a., DRL server) using Python BaseHTTPServer, same as in [11]. At each decision epoch, the DASH client sends statistical information to the DRL server, PnP-DRL derives an appropriate action based on the received statistical information. It is worth noting that for simplicity, we place the DRL server on a local machine. In practice, the DRL server can be easily deployed either locally or remotely, like in a cloud center or on a video server.



(a) On Norway-3G network (b) On AWS network (c) On Unseen network

Fig. 5: Performance of all the methods on different networks



(a) On Norway-3G network (b) On AWS network (c) On Unseen network

Fig. 6: CDF of average QoE from all methods on different networks

B. Experiment Setup

To evaluate the performance of PnP-DRL on realistic network environments, we conducted extensive experiments on two widely-used network traces, *Norway 3G dataset* [24] and *AWS trace* [30]. We applied the same pre-processing [11] to the original traces. The long-time traces are split into small traces, with each representing a 320s duration. Trivial traces whose throughput are either too small to support any available bitrate or larger than maximum bitrate are filtered out to simplify the process. We conducted experiments with both Mahimahi [31] emulation and chunk-level simulation [11] environments. For a comprehensive evaluation, we compared PnP-DRL with four popular ABR algorithms,

1) **Buffer-Based** [1]: This is a rule-based method, which directly maps the current level of buffer occupancy to a bitrate based on a pre-defined fixed rule.
2) **BOLA** [4]: It selects bitrates by Lyapunov optimization which considers the level of buffer occupancy.
3) **RobustMPC** [6]: It predicts the future throughput based on the harmonic mean of the measured throughput for the last 5 video chunks, then it selects the bitrate with the objective of maximizing the QoE metric over 5 future chunks based on the buffer occupancy and the predicted throughput. Each measured throughput is normalized by the maximum error observed in the last 5 video chunks.
4) **DRL-Based** [11]: We use an online DRL-based ABR algorithm, called *Pensieve*, which uses A3C [32] for parallel online training.

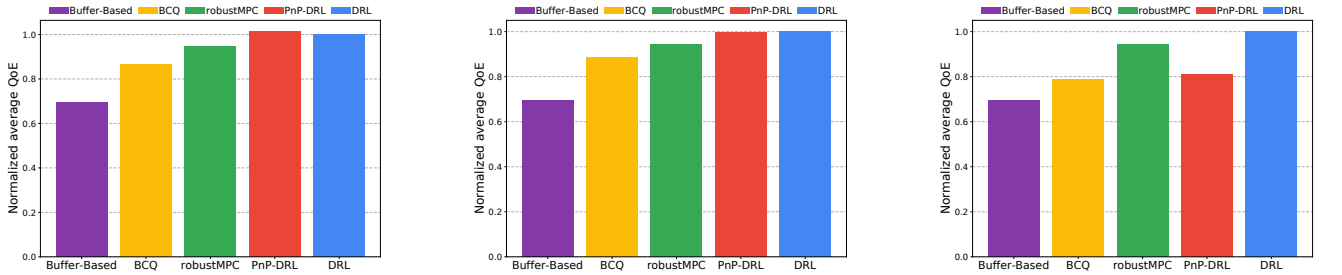
In addition, we applied the original BCQ algorithm [17] for offline training of the DRL agent.

We used the defined **reward** (e.g, *QoE value*) as the performance metric for comparisons. Before training, we first collect transitions from the existing policies on varies of network conditions. On Norway-3G and AWS traces, we totally collected 19,120, and 11,179 transition samples, respectively. We then trained the DRL agent on each trace to get a ready-to-use PnP-DRL agent. It should be noted that to evaluate the generalization of PnP-DRL, we reserved 20% of the Norway-3G traces as unseen scenarios, which means that no transition samples were collected on these traces. After offline training on the rest traces and being *pluged* to the DRL server, PnP-DRL can, promptly and seamlessly, make proper action decisions based on states of the system in these unseen scenarios.

C. Evaluation Results

The experimental results are shown in Figs. 5 - 8. Evaluating from multiple perspectives, we make the following observations.

1) Average QoE performance: From Fig. 5, we can observe that PnP-DRL significantly outperforms other non-DRL baseline ABR algorithms in terms of average QoE on all networks. Specifically, PnP-DRL achieves 834%, 74.06%, 17.65%, and 4.98% more average QoE on the Norway-3G network, compared with Buffer-Based (BB), BCQ, RobustMPC, and BOLA, respectively. It is not surprising to observe that Buffer-Based



(a) without noise in transitions (b) with 5% random noise in transitions (c) with 50% random noise in transitions

Fig. 7: Performance of all the methods with imperfect transition samples

ABR algorithm does not perform well on highly-variant cellular networks, since it makes decisions based on fixed-rule and fails to adapt to dynamic network environment. Moreover, PnP-DRL consistently performs better than vanilla BCQ on all networks, which well proved the effectiveness of our newly designed features. These results validated the effectiveness and superiority of PnP-DRL. Fig. (5c) illustrates that PnP-DRL can adapt to unknown environment, compared with other baseline solutions, with better results. Specifically, PnP-DRL achieves 355%, 67.53%, 10.68%, and 31.64% more average QoE, compared with BB, BCQ, RobustMPC, and BOLA, respectively. Last but not least, PnP-DRL delivers similar and often better QoE performance than the online DRL solution. Recall that in a typical DRL setting, the agent could progressively improve its control policy with new transition samples, which are collected continuously by interacting with the environment (a system) in a trial-and-error manner. By contrast, PnP-DRL, as a batch RL algorithm, learns the policy only from a given offline dataset, which may limit its learning ability (although it still outperforms most of the non-DRL methods). Therefore, in general, the performance of online DRL algorithms is better than batch RL algorithms. This phenomenon can also be observed in recent batch RL works [33], [34].

2) Cumulative distribution function (CDF) of QoE: The CDFs of average QoE results of algorithms are shown in Fig. 6, which provide more details about the decisions. The distribution of BCQ has long tails on negative QoE on both Norway-3G and unseen networks while RobustMPC shows similar negative skewness on AWS network. In contrast, PnP-DRL is robust to network conditions changes, and performs well on all traces, hardly ever suffering any negative QoE. As for the AWS network, because the network condition is relatively stable, we can observe from the Fig. 6b that the range of achieved average QoE under different traces is relatively small.

3) Robustness of solutions: To evaluate the robustness of PnP-DRL, we conducted experiments to see how PnP-DRL can adapt to noisy data, i.e., learning from imperfect transitions. To introduce noisy transitions, we added 5% and 50% random noise uniformly to the selected actions. The results shown in Fig. 7 proved that PnP-DRL is robust to the noisy data. Even with 50% random noise, PnP-DRL can still learn a good control policy that outperforms other baselines. Specifically, PnP-DRL achieves 17.17% and 2.83% more average QoE, compared with BB and BCQ, respectively. Both our method and BCQ learn from a generative model in a batch-constrained way, thus they can both learn a better-than-baseline policy even from noisy transitions. However, due to our particular neural network design, PnP-DRL consistently performs better under different noise levels. The DRL method shown in Fig. 7 is fully online trained and makes decision without any random noise. Therefore, online DRL is supposed to delivery a (near-)best performance, which can be regarded as upper bound results. As we can see from Fig. 7b and 7c, when there exists minor noise in the dataset, PnP-DRL can maintain the performance. However, if noise level gets significant, where the quality of collected transition samples cannot be ensured, PnP-DRL fails to learn a satisfying policy. It is worth noting that learning from imperfect dataset is still an open question in the research area of batch RL [23], [35], which is beyond the scope of this work.

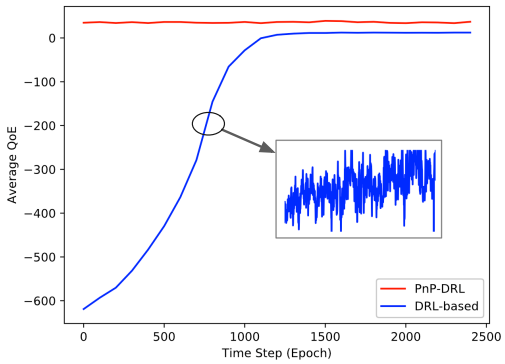


Fig. 8: Performance fluctuation during learning

4) Plug and play without learning gap: One of the most important motivations for our plug and play DRL agent is to avoid system disruption. In Fig. 8, we show that PnP-DRL provide a Plug-In-Play method for a real system, without learning gaps and system disruption after deployment. The performance of PnP-DRL is consistent and reliable. On the other hand, *Pensieve*, an online DRL solution, causes system performance variation during training. As we can see, the DRL-based method starts to explore the environment at the beginning of the training (around 1,000th time step) because of randomly initialized neural networks, thus the QoE is

fluctuating a lot. While our PnP-DRL works consistently well right after deployment, demonstrating the idea of *plug and play*. More importantly, PnP-DRL not only behaves more consistently than *Pensieve*, it also has comparable performance of average QoE Figs. 5 and 7.

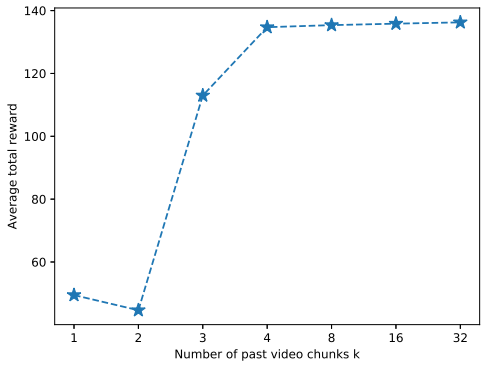


Fig. 9: The impact of historical measurements

5) Impact of historical measurements: In PnP-DRL, we propose to use an LSTM to analyze the time-series data from past k video chunks. In Fig. 9, we conducted chunk-level simulation on the FCC network traces [36] to evaluate that how many past chunks k are sufficient to represent the state space and maintain a satisfying performance. As we can see, PnP-DRL suffers from serious performance degradation when considering only a small number of past chunks, e.g., 1, 2, and 3, into the state. On the other hand, using a larger number of past chunks, e.g., 16 or 32, can provide only a marginal improvement (around 1%) while causing extra burden to data collection and offline training. Therefore, in this paper, we choose to set $k = 8$ to balance the performance and cost.

VI. Related Work

Deep Reinforcement Learning: DRL has emerged as a *de facto* approach to many complex tasks. As a pioneer work, Minh *et al.* [19] proposed DQN that leverages DNNs as function approximators to directly learn control policies from high-dimensional raw images of Atari video games. To further improve the learning ability of DRL, Hasselt *et al.* [37] proposed Double Q-learning to reduce the estimation errors introduced by DNNs. An asynchronous DRL framework (A3C) that enables parallel training for better data efficiency was presented by Mnih *et al.* [32]. Wang *et al.* [38] proposed a dueling DNN structure for DRL that work as the state value function and the state-dependent action advantage function. However, according to previous works [11], [8], it is usually not trivial to apply DRL to practical networking problems.

DRL-based Experience-Driven Networking: Recently, DRL has attracted research attention in the context of experience-driven networking because of its effectiveness and model-free property. Mao *et al.* [39] presented DeepRM, which enables a cluster system to learn the best resource scheduling policy from its control experience. [40], [41], and [7] proposed DRL-based congestion control policies that aim at reducing the end-to-end delay as well as improving throughput. Xu *et al.* [8] proposed to apply Deep Deterministic Policy Gradient (DDPG) and prioritized experience replay to solve the general traffic engineering problem. A DQN-based framework was presented by Li *et al.* [42] to solve the computation offloading and resource allocation problem in a mobile edge computing system. Wang *et al.* [43] applied DQN to solve the dynamic multichannel access problem in a wireless network particularly whose channels are correlated and system statistics is unknown. Ayala-Romero [44] presented vrAIn, a dynamic DRL-based resource controller for the virtualization of radio access network (vRAN). The results show that vrAIn can successfully derive appropriate compute and radio control actions irrespective of the platform and context.

However, all the above DRL-based networking methods are required to online interact with the system during learning, which, as we have perceived, is not realistic in practical environments.

Batch Reinforcement Learning: Batch reinforcement learning [22] has been proposed to offline train a DRL agent from a fixed batch of data, without directly interacting with the environment. It is useful in some real-world applications, particularly where data collection is dangerous or time-consuming [20]. Many existing off-policy DRL methods, such as DQN [19], double DQN [37], and DDPG [32], can work in a batch RL setting [22]. However, these methods fail to handle the extrapolation error [17], [23], which is introduced by evaluating state-action pairs that are not contained in the provided dataset. Recently, extensive research efforts have been made to eliminate the extrapolation error in batch RL. Fujomoto *et al.* [17] proposed the Batch-Constrained deep Q-learning framework (BCQ). In BCQ, the DRL agent favors more a state-action visitation that is similar to some subset of the pre-collected dataset. Based on the core idea of BCQ, Kumar *et al.* [45] presented an actor-critic algorithm, Bootstrapping Error Accumulation Reduction Q-Learning (BEAR-QL), which samples actions from a learned actor rather than a generative model. Jaques *et al.* [46] presented Way Off-Policy (WOP), which uses KL-control to penalize divergence from a pre-trained prior model of probable actions, thus reducing extrapolation error and enabling effective offline learning.

However, even though these works demonstrated that they can achieve state-of-the-art performance on many Atari game and robotic control tasks [18], it does not seem that they can be directly applied into experience-driven networking problems. As a first step, we presented two techniques to enhance the BCQ [17], [18] algorithm, LSTM-based representation network and transitions imbalanced training, which are shown to achieve better performance in a real-world application.

Dynamic Adaptive Streaming over HTTP: As video transmission has evolved to dominate today's Internet traffic in the last decade [47], Dynamic Adaptive Streaming over HTTP (DASH) has emerged as a dominant standard for video transmission. Through Adaptive BitRate (ABR) algorithms, DASH enables the video server to optimize the video quality as well as improving user's QoE. Most of the existing ABR algorithms are rule-based (e.g., Buffer-based [1]) or optimization-based

(e.g., BOLA [4], MPC [6]) solutions. They rely on accurate mathematical system models, and typically fail to adapt to the dynamics of networking systems, consequently, are usually unavailable in complicated real systems. Recent breakthrough of DRL provides a promising alternative for enabling model-free experience-driven ABR for DASH. Gadaleta *et al.* [10] proposed D-DASH, a framework that applies deep Q-learning to optimize the QoE for DASH. Mao *et al.* [11] presented Pensieve, a system that learns ABR without relying on any pre-programmed control rules or explicit assumptions about the networking system. Although rule-based [1] or optimization-based [4], [6] ABR algorithms, which do not need training, are easy to deploy, they are usually limited by possible inaccurate system models. DRL-based ABR algorithms, including both online and batch RL (e.g., our PnP-DRL), learn the best control policy from the environment without relying on any assumed system model. More importantly, compared with online DRL ABR algorithms [10], [11], PnP-DRL can learn from a fixed dataset without further interactions with environment (running system). Such characteristic helps to avoid the interruption of running systems, while still achieving a comparable or even better performance.

While the goal of this paper is not to propose a new or better ABR algorithm, we used the DASH problem as a study case in order to illustrate that online DRL may have side-effect to real-world networking systems. We proposed to train a plug and play DRL agent with pre-collected samples, which can outperform traditional ABR algorithms.

VII. Conclusion

Though DRL has emerged as a *de facto* approach to many complex experience-driven networking problems, practical deployment of these DRL policies faces realistic challenges. In this work, we first analyzed and demonstrated that online DRL solutions could be unstable and even dangerous to functioning systems. We then proposed to leverage batch RL, such as BCQ, to offline train a *plug and play* DRL agent, which can be applied into system seamlessly and take effect promptly without further interactions with environment. With two proposed new techniques, *LSTM-based representation network* and *transitions imbalanced training*, PnP-DRL improved BCQ and empower offline DRL agent to be plug and play for experience-driven networking. Extensive experiments manifested that our plug and play DRL agent behaves consistently without any disruption of the system, and outperforms baseline ABR algorithms while achieving comparable performance to the online DRL solutions.

References

[1] T. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, A buffer-based approach to rate adaptation: Evidence from a large video streaming service, *ACM SIGCOMM'14*, pp. 187–198.

[2] Open Shortest Path First (OSPF), https://en.wikipedia.org/wiki/Open_Shortest_Path_First

[3] R. Z. Shen, Valiant Load-Balancing: building networks that can support all traffic matrices, *Algorithms for Next Generation Networks*, 2010.

[4] K. Spiteri, R. Urgaonkar, and R. Sitaraman, BOLA: Near-optimal bitrate adaptation for online videos, *IEEE INFOCOM'16*, pp. 1–9.

[5] D. Xu, M. Chiang and J. Rexford, Link-state routing with hop-by-hop forwarding can achieve optimal traffic engineering, *IEEE/ACM Transactions on networking*, Vol. 19, No. 6, 2011, pp. 1717–1730.

[6] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over HTTP, *ACM SIGCOMM'15*, pp. 325–338.

[7] N. Jay, N. Rotman, B. Godfrey, M. Schapira, and A. Tamar, A deep reinforcement learning perspective on internet congestion control, *ICML'19*, pp. 3050–3059.

[8] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu and D. Yang, Experience-driven networking: a deep reinforcement learning based approach, *IEEE INFOCOM'18*, pp. 1871–1879.

[9] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar. Learning to route, ACMWorkshop on HotNets, 2017, pp. 185–191.

[10] M. Gadaleta, F. Chiariotti, M. Rossi, and A. Zanella, D-DASH: a deep Q-learning framework for DASH video streaming, *IEEE Transactions on Cognitive Communications and Networking*, Vol. 3, No. 4, 2017, pp. 703–718.

[11] H. Mao, R. Netravali, and M. Alizadeh, Neural adaptive video streaming with pensieve, *ACM SIGCOMM'17*, pp. 197–210.

[12] M. Van Der Schaar, D. S Turaga, and R. Wong, Classification-based system for cross-layer optimized wireless video transmission, *IEEE Transactions on Multimedia*, Vol. 8, No. 5, 2006, pp. 1082–1095.

[13] Y. Sun, X. Yin, J. Jiang, V. Sekar, F. Lin, N. Wang, T. Liu, and B. Sinopoli, CS2P: Improving video bitrate selection and adaptation with data-driven throughput prediction, *ACM SIGCOMM'16*, pp. 272–25.

[14] M. Wang, Y. Cui, X. Wang, S. Xiao, and J. Jiang, Machine learning for networking: Workflow, advances and opportunities, *IEEE Network*, Vol. 32, No. 2, 2017, pp. 92–99.

[15] S. Krishnan, and R. Sitaraman, Video stream quality impacts viewer behavior: Inferring causality using quasi-experimental designs, *IEEE/ACM Transactions on Networking*, Vol. 21, No. 6, 2013, pp. 2001–2014.

[16] https://www.conviva.com/research/consumer-survey-report-2015-how-consumers-judge-their-viewing-experience/

[17] S. Fujimoto, D. Meger, and D. Precup, Off-Policy deep reinforcement learning without exploration, *ICML'19*, pp. 2052–2062.

[18] S. Fujimoto, E. Conti, M. Ghavamzadeh, and J. Pineau, Benchmarking batch deep reinforcement learning algorithms, *arXiv: 1910.01708*, 2019.

[19] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg and D. Hassabis, Human-level control through deep reinforcement learning, *Nature*, Vol. 518, No. 7540, 2015, pp. 529–533.

[20] T. Li, Z. Xu, J. Tang, and Y. Wang, Model-free control for distributed stream data processing using deep reinforcement learning, *Proceedings of the VLDB Endowment*, Vol. 11, No. 6, 2018, pp. 705–718.

[21] S. Gu, T. Lillicrap, I. Sutskever and S. Levine, Continuous deep Q-Learning with model-based acceleration, *ICML'16*, pp. 2829–2838.

[22] S. Lange, T. Gabel, and M. R. Miller, Batch reinforcement learning, *Reinforcement Learning*, Springer, 2012.

[23] S. Levine, A. Kumar, G. Tucker, and J. Fu, Offline reinforcement learning: tutorial, review, and perspectives on open problems, *arXiv: 2005.01643*, 2020.

[24] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, Commute path bandwidth traces from 3G Networks: analysis and applications, *ACM MMSys'13*, pp. 114–118.

[25] T. Stockhammer, Dynamic adaptive streaming over HTTP – standards and design principles, *ACM Conference on Multimedia System*, pp. 133-144, 2011.

[26] J. Johnson, and T. Khoshgoftaar, Survey on deep learning with class imbalance, *Journal of Big Data*, Vol. 6, No. 1, 2019, pp. 27.

[27] A. Paszke, S. Gross, S. Chintala, and *et al.*, Automatic differentiation in pytorch, 2017.

[28] V. Nair, and G. Hinton, Rectified linear units improve restricted boltzmann machines, *ICML'10*.

[29] D. Kingma and J. Ba, Adam: a method for stochastic optimization, *ICLR'15*.

[30] Y. Yan, J. Ma, G. Hill, D. Raghavan, R. Wahby, P. Levis, and K. Winstein, Pantheon: the training ground for Internet congestion-control research, *USENIX ATC'18*, pp. 731–743.

[31] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan, Mahimahi: accurate record-and-replay for HTTP, *USENIX ATC'15*, pp. 417–429.

[32] V. Mnih, *et al.*, Asynchronous methods for deep reinforcement learning, *ICML'2016*, pp. 1928–1937.

[33] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine, D4rl: Datasets for deep data-driven reinforcement learning, *arXiv: 2004.07219*, 2020.

[34] C. Gulcehre, Z. Wang, A. Novikov, T. Paine, S. Gómez, K. Zolna, R. Agarwal, J. S Merel, D. J Mankowitz, C. Paduraru, G. Dulac-Arnold. RL Unplugged: A collection of benchmarks for offline reinforcement learning, *NeurIPS'20*.

[35] Y. Wu, and N. Charoenphakdee, H. Bao, V. Tangkaratt, and M. Sugiyama, Imitation learning from imperfect demonstration, *ICML'19*, pp. 6818–6827.

[36] Federal Communications Commission: https://www.fcc.gov/reports-research/reports/

[37] H. v. Hasselt, A. Guez, and D. Silver, Deep reinforcement learning with double Q-learning, *AAAI'16*, pp. 2094–2100.

[38] Z. Wang, T. Schaul, M. Hessel, H. Van, M. Lanctot and N. De Freitas, Dueling network architectures for deep reinforcement learning, *ICML'16*, pp. 1995–2003.

[39] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, Resource management with deep reinforcement learning, *HotNets'16*, pp. 50–56.

[40] S. Emara, B. Li, and Y. Chen, Eagle: Refining congestion control by learning from the experts, *IEEE INFOCOM'20*, pp. 676–685.

[41] Z. Xu, J. Tang, C. Yin, Y. Wang, and G. Xue, Experience-driven congestion control: When multi-path TCP meets deep reinforcement learning, *IEEE JSAC*, vol. 37, no. 6, pp. 1325–1336, 2019.

[42] J. Li, H. Gao, T. Lv, and Y. Lu, Deep reinforcement learning based computation offloading and resource allocation for MEC, *IEEE Wireless Communications and Networking Conference (WCNC)*, 2018, pp. 1–6.

[43] S. Wang, H. Liu, P. H. Gomes, and B. Krishnamachari, Deep reinforcement learning for dynamic multichannel access in wireless networks, *IEEE Transactions on Cognitive Communications and Networking*, vol. 4, no. 2, pp. 257–265, 2018.

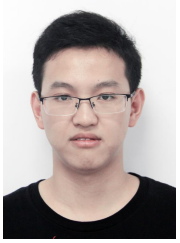
[44] J. Ayala-Romero, A. Garcia-Saavedra, M. Gramaglia, X. Costa-Perez, A. Banchs, and J. Alcaraz, vrAIn: A deep learning approach tailoring computing and radio resources in virtualized RANs, *ACM MobiCom'19*, pp. 1–16.

[45] A. Kumar, J. Fu, M. Soh, G. Tucker, and S. Levine, Stabilizing off-policy q-learning via bootstrapping error reduction *NeurIPS'19*, pp. 11784–11794.

[46] N. Jaques, A. Ghandeharioun, J. Shen, C. Ferguson, A. Lapedriza, N. Jones, S. Gu, and R. Picard, Way off-policy batch deep reinforcement learning of implicit human preferences in dialog, *arXiv: 1907.00456*, 2019.

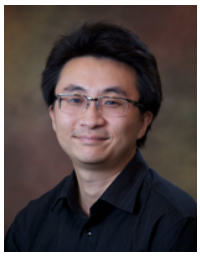
[47] Index, Cisco Visual Networking, Global mobile data traffic forecast update, 2016–2021, *White Paper*, Vol. 7, 2017.

Zhiyuan Xu is currently pursuing the Ph.D. degree at the Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY, USA. He received the B.E. degree in School of Computer Science and Engineering from University of Electronic Science and Technology of China, Chengdu, China, in 2015. He was an exchange student in 2013 at Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan. He was a visiting student in 2015 at Dalhousie University, Halifax, NS, Canada. His current research interests include deep reinforcement learning, communication networks, and edge computing.



Kun Wu received his B.S. degree from Beijing Institute of Technology in 2017. He is currently working towarding his Ph.D degree in the Department of Electrical Engineering and Computer Science at Syracuse University. His research interests include Machine Learning and Computer Vision.

Weiyi (Max) Zhang is a Senior Consultant at Network Evolution Strategies, LLC, Holmdel, NJ, USA. His research interests include reinforcement learning on network planning and optimization, SDN and network function virtualization for carrier networks, network traffic demand forecast and analysis, 5G wireless broadband strategic design. He has published more than 100 refereed papers in his research areas. He received Best Paper Awards from IEEE ICNP'2017, ICC'2014, GLOBECOM'2007.



Jian Tang (F19) is a Professor in the Department of Electrical Engineering and Computer Science at Syracuse University, an IEEE Fellow and an ACM Distinguished Member. He received his Ph.D degree in Computer Science from Arizona State University in 2006. His research interests lie in the areas of AI, IoT, Wireless Networking, Mobile Computing and Big Data Systems. Dr. Tang has published over 160 papers in premier journals and conferences. He received an NSF CAREER award in 2009. He also received several best paper awards, including the 2019 William R. Bennett Prize and the 2019 TCBD (Technical Committee on Big Data) Best Journal Paper Award from IEEE Communications Society (ComSoc), the 2016 Best Vehicular Electronics Paper Award from IEEE Vehicular Technology Society (VTS), and Best Paper Awards from the 2014 IEEE International Conference on Communications (ICC) and the 2015 IEEE Global Communications Conference (Globecom) respectively. He has served as an editor for several IEEE journals, including IEEE Transactions on Big Data, IEEE Transactions on Mobile Computing, etc. In addition, he served as a TPC co-chair for a few international conferences, including the IEEE/ACM IWQoS'2019, MobiQuitous'2018, IEEE iThings'2015. etc.; as the TPC vice chair for the INFOCOM'2019; and as an area TPC chair for INFOCOM 2017-2018. He is also an IEEE VTS Distinguished Lecturer, and the Chair of the Communications Switching and Routing (CSR) Technical Committee of IEEE ComSoc.

Yanzhi Wang is currently an assistant professor in the Department of Electrical and Computer Engineering at Northeastern University. He has received his Ph.D. Degree in Computer Engineering from University of Southern California (USC) in 2014, and his B.S. Degree with Distinction in Electronic Engineering from Tsinghua University in 2009. Dr. Wang's current research interests are the energy-efficient and high-performance implementations of deep learning and artificial intelligence systems. Besides, he works on the application of deep learning and machine intelligence in various mobile and IoT systems, medical systems, and UAVs, as well as the integration of security protection in deep learning systems. His works have been published in top venues in conferences and journals (e.g. ASPLOS, MICRO, HPCA, ISSCC, AAAI, ICML, ICLR, ECCV, ACM MM, CCS, VLDB, FPGA, DAC, ICCAD, DATE, LCTES, INFOCOM, ICDCS, Nature SP, etc.), and have been cited for around 4,000 times according to Google Scholar. He has received four Best Paper Awards, has another seven Best Paper Nominations and two Popular Papers in IEEE TCAD. His group is sponsored by the NSF, DARPA, IARPA, AFRL/AFOSR, and industry sources.

Guoliang Xue (F11) is a Professor of Computer Science and Engineering at Arizona State University. He received the Ph.D degree in Computer Science from the University of Minnesota in 1991. His research interests span the areas of QoS provisioning, machine learning, wireless networking, network security and privacy, crowdsourcing and network economics, Internet of Things, smart city and smart grids. He has published over 300 papers in these areas, many of which in top conferences such as INFOCOM, MOBICOM, NDSS and top journals such as IEEE/ACM ToN, IEEE JSAC, IEEE TDSC, and IEEE TMC. He has received the IEEE Communications Society William R. Bennett Prize in 2019 (best paper award for IEEE/ACM TON and IEEE TNSM in the previous three years). He was a keynote speaker at IEEE LCN'2011 and ICNC'2014. He was a TPC Co-Chair of IEEE INFOCOM'2010 and a General Co-Chair of IEEE CNS'2014. He has served on the TPC of many conferences, including ACM CCS, ACM MOBIHOC, IEEE ICNP, and IEEE INFOCOM. He served on the editorial board of IEEE/ACM Transactions on Networking and the Area Editor of IEEE Transactions on Wireless Communications, overseeing 13 editors in the Wireless Networking area. He is an IEEE Fellow, and the Steering Committee Chair of IEEE INFOCOM.