Cracking Open the Black Box: What Observations Can Tell Us About Reinforcement Learning Agents

Arnaud Dethise
KAUST

Marco Canini
KAUST

Srikanth Kandula
Microsoft

ABSTRACT

Machine learning (ML) solutions to challenging networking problems, while promising, are hard to interpret; the uncertainty about how they would behave in untested scenarios has hindered adoption. Using a case study of an ML-based video rate adaptation model, we show that carefully applying interpretability tools and systematically exploring the model inputs can identify unwanted or anomalous behaviors of the model; hinting at a potential path towards increasing trust in ML-based solutions.

CCS CONCEPTS

• **Information systems** → Multimedia streaming; • **Networks** → Network resources allocation; • **Computing methodologies** → Reinforcement learning.

KEYWORDS

explainable machine learning, post-hoc explanations, feature analysis, neural adaptive video streaming

ACM Reference Format:
Arnaud Dethise, Marco Canini, and Srikanth Kandula. 2019. Cracking Open the Black Box: What Observations Can Tell Us About Reinforcement Learning Agents. In *NetAI '19: ACM SIGCOMM 2019 Workshop on Network Meets AI & ML, August 23, 2019, Beijing, China.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3341216.3342210

1 INTRODUCTION

Machine learning (ML) based solutions have recently been developed for several problems in networked systems such as resource allocation [10], routing [16], video rate selection [11] and congestion control [3]. Many of these problems do not have tractable optimal solutions. A key advantage of learning-based solutions is that they can be attuned to specific problem instances; for example, a video rate adaptation method that is tuned specifically for Netflix clients in the United States may perform better than a generic method; or a cluster job scheduler trained for the jobs at a particular company may perform better than off-the-shelf generic heuristics. Moreover, learning can avoid manual effort to tune heuristics. Current results, while early, are laudable because they achieve improvements even on problems that have been studied intensively. This trend may accelerate and some authors already call for self-driving networks [4, 7].

Yet, there is a general fear that ML systems are black boxes: closed systems that receive an input, produce an output, and offer no clue why. This creates uncertainty about why these systems work, whether they will continue to work in conditions that are different from those seen during training or whether they will fall off performance cliffs. These issues are not specific to the networking community and it is possible that the lack of interpretability and behavioural uncertainity is a reason why ML-based solutions for systems problems have thus far not been widely adopted [5, 13, 18].

Some researchers are working towards models that are easy-to-interpret [12, 17]; however, current interpretable solutions have lower performance and longer training times than black box techniques. For the non-interpretable models, some recently developed techniques can analyze the reasons behind the decisions taken by a model [2, 8, 14]. We propose to use these tools to examine a model's decisions on various inputs (e.g., real network traces) and we compare the model's outputs with the outputs expected based on domain knowledge.

In this paper, we use this method to understand the behavior of Pensieve [11], a model that targets the video bit rate adaptation problem and is built using a popular reinforcement learning approach. Our main contribution is to show what kind of information can be extracted by carefully inspecting behavior-revealing data and using post-hoc explanations. Our analysis also builds intuition on the benefits of this approach by showing the performance and pitfalls of the Pensieve agent. Finally, we raise some additional concerns when applying interpretability tools to explain ML-based solutions for networked systems.

Our findings for Pensieve can be classified into three types: observations about the agent's decisions (Section 4), understanding how individual input features have contributed to these decisions (Section 5) and the broader relationship between feature values and decisions (Section 6).

Our case study reveals several interesting aspects; we highlight two of them. (1) The Pensieve agent bases its decisions on very few of the inputs. That is, using just a handful of inputs leads to decisions that are nearly as good as the 25 features used by the authors of Pensieve; fewer features would simplify training and lead to a more succinct solution. (2) The Pensieve agent rarely uses two out of the six bit rates available (e.g., rates 1200Kbps and 2850Kbps are almost never used); this happens even though training data contains a large fraction of traces for which using these rates is appropriate. We find that the reason is that time-multiplexing between rates on either side of the ignored rates leads to a reasonable QoE.

We take care to mention that we fully expect such anomalies to be discovered in other recent ML-based solutions; we believe our method can be generalized to several of these. We chose Pensieve

NetAI '19, August 23, 2019, Beijing, China

ACM ISBN 978-1-4503-6872-8/19/08…$15.00
https://doi.org/10.1145/3341216.3342210

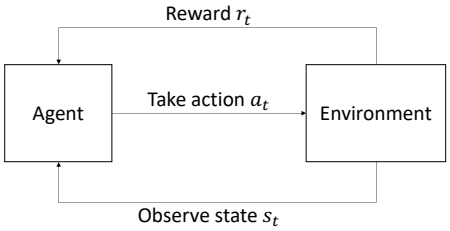

Figure 1: A typical reinforcement learning model.

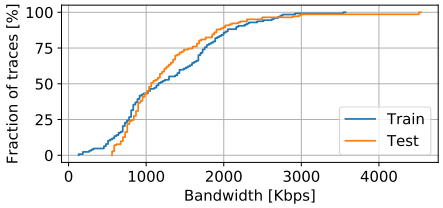

Figure 2: CDF of the network bandwidth in the testing and training sets.

primarily because the authors kindly shared their code and model. The goal with this example is to illustrate the value in understanding ML models beyond just looking at their bottom-line performance. Our hope with this paper is to initiate a discussion around the importance and benefits of doing so.

2 BACKGROUND

To explore how to understand the behaviour of a trained neural network, we focus on the Pensieve agent [11]. Here, we describe the RL architecture and the video rate selection problem considered by Pensieve.

2.1 Reinforcement Learning

A reinforcement learning approach considers an *agent* interacting with an *environment*. At each time step t, the agent selects an action a_t based on the current state of the environment s_t; applying the action changes the environment and yields a reward r_t. The goal of training is to learn to select actions that maximize expected rewards. Figure 1 depicts the typical RL architecture.

In some cases, such as with Pensieve, the RL model is trained using a simulator of the environment by replaying traces; Pensieve replays traces that contain available bandwidth along a network path and the simulator mimics streaming video at different bit rates. Pensieve is trained to maximize the *discounted sum* of future rewards, $\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$ where $\gamma \in [0, 1)$ is the discount factor.

The RL approach is popular to train agents in networked systems because such agents integrate well with decision-making in existing software. These systems typically record a lot of information about the environment. Neural networks can identify useful features from that information even when given many potential features. Another benefit of using RL for systems is that it is intractable to find optimal decisions; by using the instantaneous performance of the system (e.g. throughput, latency, utilization, quality of experience) as a reward, the agent when trained can maximize performance metrics.

2.2 Adaptive Bit Rate Selection in Pensieve

In video streaming services, videos are typically divided into fixed-length *chunks* that are encoded at multiple bit rates. When the client finishes downloading a chunk, it chooses the bit rate at which to download the next chunk; higher bit rates increase perceived quality but consume more network bandwidth.

The goal of adaptive bit rate selection in Pensieve is to maximize a QoE function, defined in [19] as:

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

where N is the total number of chunks, $q(R_n)$ is the perceived quality for the selected bit rate R_n, μT_n is a penalty for time spent re-buffering, and $|q(R_{n+1})-q(R_n)|$ is a penalty for non-smooth video playback, incurred when the bit rate changes. In all our experiments, we use the linear reward function $q(R_n) = R_n$.

The inputs for the agent consist of 25 features that include the current buffer duration, the inferred network throughput and download time of the past 8 chunks, the sizes available for the next chunks, the number of remaining chunks until the end of the video and the previously selected bit rate. Chunks are 4-second long and the total length of a video is 48 chunks.

The action space consists of the following 6 bit rates in Kbps: 300, 750, 1200, 1850, 2850 and 4300. Hence, the agent outputs a vector of 6 probabilities, each value being the probability to select a specific bit rate. The client decides the next-chunk's bit rate by sampling based on these probabilities.

3 EXPERIMENTAL METHODOLOGY

Before we delve into the experimental results of our analysis, we clarify how we run our experiments and collect measurements. Unless otherwise noted, we run the pre-trained agent on every network trace in the testing set provided by the Pensieve authors. At every decision step, we collect the output probabilities of each decision. When presenting statistical results, we make use of all decisions from every trace as samples.

We contrast the composition of traces in the training and testing sets to help understand whether some behaviors of the agent are due to insufficient training, due to an incomplete training dataset or due to some other reason. Figure 2 shows the distribution of the average available bandwidth in training and testing sets; we observe that the distributions are similar. The largest difference is at the 70^{th} percentile, which is 1400Kbps in the testing set and 1700Kbps in the training set. However, the minimum and maximum average available bandwidth in the testing set are higher (550 and 4550Kbps, respectively) than in the training set (1200 and 3550Kbps, respectively).

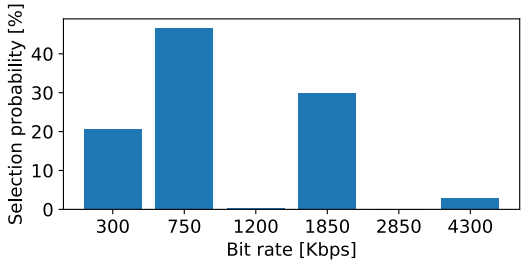

Figure 3: Frequency with which each possible bit rate is selected over the testing set.

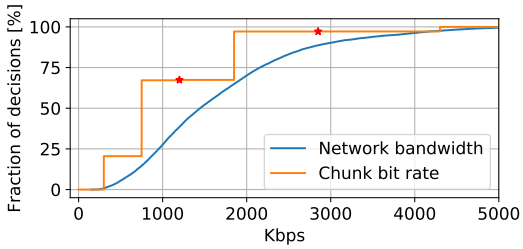

Figure 4: CDF of the available bandwidth at each decision step in the testing set (blue) and the bit rate selected for the next chunk by Pensieve (orange).

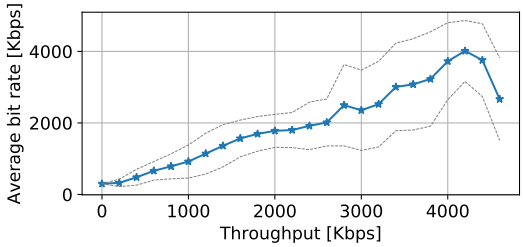

Figure 5: Average bit rate selected by Pensieve for various measured throughput values. Dashed lines show the standard deviation.

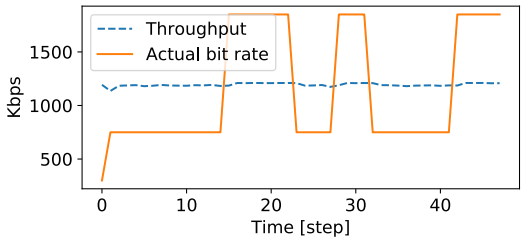

Figure 6: Behavior of Pensieve for a synthetic network trace with constant available bandwidth of 1200Kbps.

4 BREAKDOWN OF DECISIONS

We start by showing how basic statistics of the outputs returned by the agent at each decision step allows us to validate whether the decision space chosen by the model's creator has been learned correctly by the neural network. We analyze the relation between throughput and decision. Then, we show that Pensieve uses just four out of the six bit rates available and we quantify the effects of this behavior on the QoE.

Probability of decisions. Figure 3 shows the average probability of selecting each bit rate.[1] This figure shows that the 750Kbps bit rate is the most likely to be chosen (47%), while the best quality (4300Kbps) is selected only 3% of the time. The most interesting observation is that *the intermediate bit rates 1200Kbps and 2850Kbps are almost never selected.*

We verify that the characteristics of the testing set do not include a bias for not selecting bit rates 1200 and 2850Kbps. Figure 4 shows the distribution of the available bandwidth in the traces at each decision (in blue) versus the bit rate selected for the chunk (in orange). Ideally, we want the orange line to be as close as possible to the blue one, so that the selected bit rate closely matches the available bandwidth. We see that this is not the case, and the ignored bit rates mentioned above (marked with red stars) create a significant gap between the two lines.

Probability of bit rate across throughput. We now breakdown the agent's bit rate selections by throughput. Specifically, we consider ever decision made and the measured throughput (aggregated in 200Kbps bins) when that decision was made. Figure 5 reports the average selected bit rate. We observe that even though the 1200Kbps bit rate is almost never selected, the average bit rate is still close to 1200Kbps when the throughput is close to this value. To illustrate how the agent behaves in this regime, we use a synthetic trace where the available bandwidth is kept constant at 1200Kbps. The bit rate decision over time is shown in Figure 6.

Empirical policy and implications on QoE. Based on the above observations, we can better make sense of the agent's policy: *the Pensieve model has learned to use only four out of the six available bit rates, and will multiplex between multiple bit rates to get closer to the actual bandwidth.* A definitive answer as to why this particular policy was learned is outside the scope of this paper. It is possibly due to regularization penalties used by policy gradient methods that force the model to prefer using fewer actions to reduce the entropy. Next, we measure the loss in video quality (reward) due to ignoring these bit rates.

Assuming that every bit rate is an admissible decision, Equation 1 shows that the optimal average reward per decision requires selecting the bit rate B equal to the available bandwidth, and $QoE = \frac{1}{N}\sum_1^N q(B) = B$. To quantify the optimality gap due to using a finite number of bit rates (recall, Pensieve effectively uses four bit rates), for a given bit rate B, we use a reward loss metric defined as the average difference between the optimal reward (introduced above) and the average Pensieve reward calculated over a synthetic trace with constant bit rate B and 4800 decision steps.

Figure 7 shows the reward loss for $B \in [300, 4300]$Kbps in 50Kbps increments. We observe that even though Pensive ignores the

[1] That is, the average over all decisions in the testing set of the probability of selecting each bit rate.

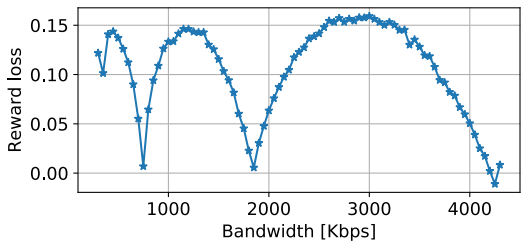

Figure 7: Difference between the optimal reward that can be achieved assuming the agent is able to choose any possible bit rate and the Pensieve agent.

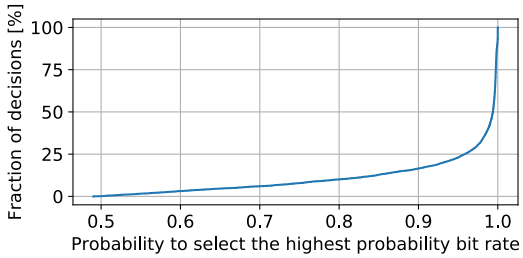

Figure 8: Confidence of the agent in its most likely decision.

1200Kbps and 2850Kbps bit rates, the reward loss when the available bandwidth is near these values is similar to the reward loss at other bandwidth values. The reward loss is always below 0.15. Thus, we conjecture that the training process chooses to ignore these bit rates because even ignoring them results in no worse loss than is achieved by just quantizing the rates.

Decision confidence. Since the agent's output is a vector of probabilities (one per bit rate), the actual bit rate is chosen randomly (weighted by bit rate probabilities) unless the probability of all but one bit rates is 0. We analyze the values of these probabilities to find how much randomness is present in the agent's decisions. In particular, we consider for each decision the highest probability among the bit rates. We call decision confidence the probability for the bit rate with highest probability and say that confidence is high when this probability is high. Figure 8 shows that 77% of the decisions have a high confidence (a probability higher than 0.95), 90% of the decisions have confidence higher than 0.8, and less than 1% of the decisions have confidence below 0.5. This reveals that most decisions are not random and ties are almost always between no more than two bit rates.

5 CONTRIBUTION OF EACH FEATURE

To further explain the Pensieve model, we seek to understand how the agent uses input features. We use an explainability tool called LIME [14] which builds a linear approximation of the model around a single decision to compute how much each feature contributes to the decision. A large contribution (or simply weight) for a feature indicates that small variations in that feature's value are more likely to change the decision; a small weight means that even large changes to a feature's value will likely not change the decision.

Rank	Feature	Weight
1	Previous bit rate	0.377
2	Buffer	0.184
3	Throughput [T]	0.154
4	Throughput [T-1]	0.028
5	Download time [T-1]	0.018
6	Download time [T]	0.018
7	Throughput [T-7]	0.017

Table 1: Normalized average weight of individual features; i.e., their extent of contribution to decisions.

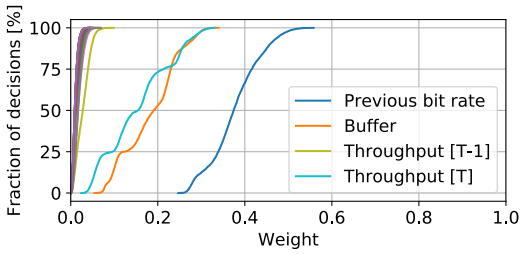

Figure 9: CDF of the individual weight of each feature on the testing dataset.

Weights are computed for each bit rate and can be positive (if changing the value decreases the probability of the chosen bit rate) or negative (if changing the value increases the probability). To account for how much the agent "*pays attention to*" each feature, we summed the absolute weight value across all classes and normalized the result so that the sum of the weight of all features is 1. Table 1 shows the *average* weight of the top seven features across all decisions. It shows that the previous bit rate (which is used to ensure smoothness), the current amount of data in the buffer and the most recent throughput value are the most important features on average, accounting for 71.5% of the weight.

One question is how often the features that are not part of the top three have an impact on the decision. To analyze this, Figure 9 shows the distribution of the weight of each feature across all decisions. We observe that only four of the 25 features (shown in the legend) have a weight above 0.07. We note that LIME is a local approximation of the model and can be noisy.

To verify that features that have a small weight as computed by LIME actually have almost no impact on a decision, we re-ran experiments wherein we replace the value of each of these features with a dummy value that is equal to the testing dataset's average value for that feature. We did this because excluding the features altogether would require retraining a new model. Our approach effectively "hides" the value of the low weight features from the model. We observe that the impact on the model's decisions is small. Figure 10 shows the distribution of the agent's QoE when all but the top k features are hidden. This figure shows that by removing all features except for the top three (green line), the performance of the agent is very close to that using all features (black line). This suggests that the Pensieve model could be simplified to take fewer inputs which, in turn, reduces its complexity and training time.

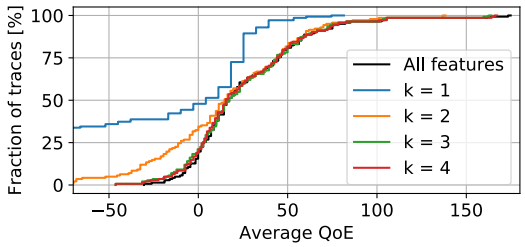

Figure 10: CDF of the average QoE achieved by hiding all but the top k features.

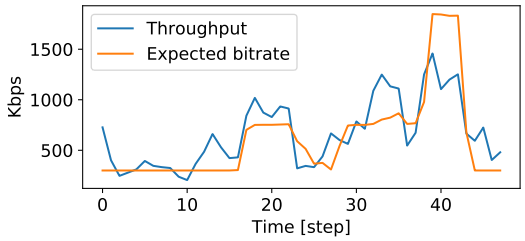

Figure 11: An example trace of how the decision changes when the network throughput varies.

6 EFFECTS OF INPUTS ON DECISIONS

We have shown that different features have different weights (i.e., contributions) to the decisions of the Pensieve agent. We now explore how the actual input values affect the decision. From a macroscopic perspective, we first show that there exists a clear relation (as expected) between the network throughput and the selected bit rate. Then, at a microscopic level, we analyze how the top three features individually affect the decision.

Perhaps the most important trait of an adaptive bit rate agent is that it adapts to varying bandwidth conditions. This is the expected behavior and we seek to confirm it experimentally. We analyze the temporal behavior of the agent and observe that the expected bit rate rises and falls with the network throughput. Figure 11 shows an example execution of the agent on a single trace. The expected bit rate, here, is the weighted average bit rate, weighted by the probability with which each bit rate was to be selected based on the decision of the Pensieve agent.

To understand the individual effect of a feature's value on the decision, we draw scatter plots relating the value of a feature and its corresponding weight to the decision as estimated by LIME. In each scatter plot, one point corresponds to one decision. Here, we focus on the three features that have a high weight in the decisions.

Figure 12 shows the weight associated for different values of the 'amount of data in the buffer' feature. Note that four ranges of feature values (0–12 seconds of buffered video, 12–16 seconds, 16–20 and more than 20 seconds of buffered video) have distinct weights; the additional variation in each range is at least partially due to the LIME approximation. This behavior can be explained as follows: when the buffer is almost empty, the agent would switch to a lower bit rate to fill it, and when the buffer is almost full, the agent would switch to a higher quality to improve the QoE. The

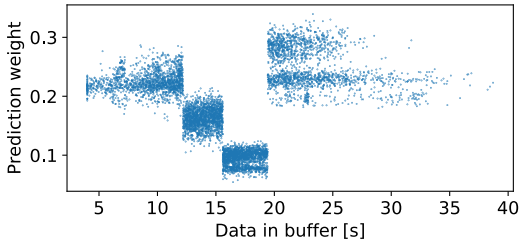

Figure 12: Relation between buffer value and weight.

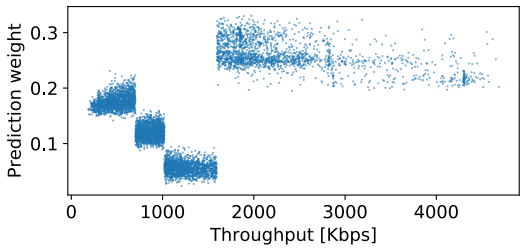

Figure 13: Relation between throughput value and weight.

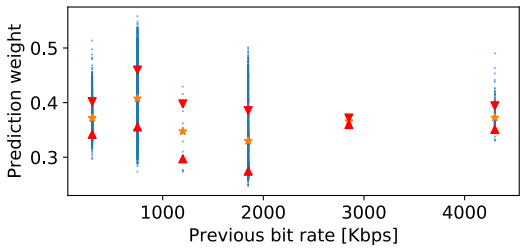

Figure 14: Relation between previous bit rate and weight. The orange star is the average, the red triangles show the standard deviation.

"sweet spot" in buffer size, as chosen by Pensieve, appears to be 16–20 seconds of video in the buffer; here, the buffer length has only a little impact on the decision.

Figure 13 shows the weight associated for different values of the 'throughput' feature. While the behavior appears similar to that in Figure 12, it is not clear why Pensive behaves in this way. Specifically, we would expect that the agent would pay as much attention to any throughput value, including those between 1000Kbps and 1600Kbps. The weight may be small here because the agent has a default assumption that throughput is in this range and only other throughput values have an effect on the decision; we are not yet able to confirm this hypothesis.

Figure 14 shows the weight for different values of the 'previous bit rate' feature (the possible values are discrete, because there are only six possible bit rates). We note that the weight is higher for 300Kbps and 750Kbps than for 1850Kbps. This is possibly because the 1850 Kbps bit rate is used for a wide range of throughput (between 1300Kbps and 3000Kbps).

Finally, Figures 15, 16 and 17 show the relationship between feature values and the expected bit rate. Here, horizontal clusters

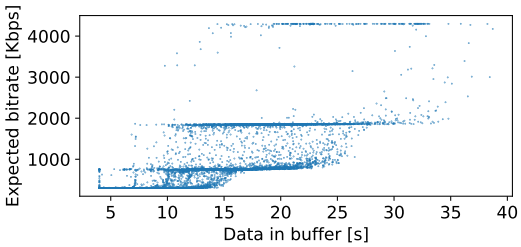

Figure 15: Relation between buffer value and chosen bit rate.

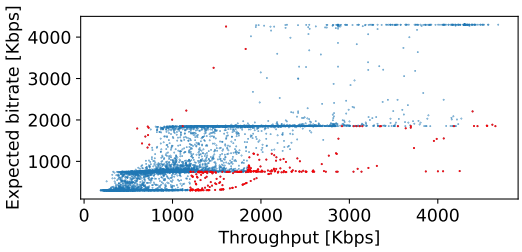

Figure 16: Relation between throughput value and chosen bit rate.

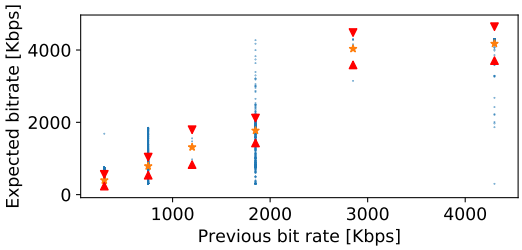

Figure 17: Relation between previous bit rate and chosen bit rate. The orange star is the average, the red triangle shows the standard deviation.

are caused by decisions where the probability of a specific bit rate is high. As expected, a large buffer and a high throughput correlate to selecting a higher bit rate. In Figure 16, we highlight points (in red) for which the difference between throughput and bit rate is greater than the gap between available bit rates. These points are worth further investigation as we will discuss in the next section.

7 LIMITATIONS

We have shown that one can understand the decisions of Pensieve to some degree despite the "black box" nature of the model that was learnt using RL. Here, we highlight additional challenges that remain and also discuss future directions for research in interpretable RL for networked systems.

The problem of explainable AI [5, 18] has become popular in the ML community. Today's solutions such as saliency maps [15], post-hoc explanations [2, 14] and interpretable models [17] fail to address some specific concerns when using models learnt using RL in networked systems. In particular, it is unclear how to combine them with the domain knowledge of system experts. Also, these solutions tend to explain individual decisions rather than the entire policy or explain where exactly a sequence of decisions may go wrong.

One method that results in models that are easy to interpret is to restrict the models to be linear regressions over input features. However, linear models might perform poorly in specific applications; indeed, our preliminary tests in the context of adaptive bit rate showed worsened QoE.

We list below some open issues towards increasing trust in RL-based solutions. At a high level, these challenges are questions that are of interest to system operators which cannot be answered by only looking at the inputs and outputs of an RL agent.

Finding outliers. A key concern about ML solutions is whether they will fall off performance cliffs. One approach to detect performance cliffs is to find outliers in the decisions: if a small change in the input leads to a significant change in the decision, it is possible that the decision may lead to a sizable performance degradation. As an example, consider the hypothetical case where the agent over-reacts to a relatively small change in measured throughput and seeks to increase bit rate to a level that cannot sustain uninterrupted video streaming. We note that finding such outliers is difficult because the input space can be very large and impractical to fully explore.

Ensuring stability and adaptation. It is reasonable to require that an RL agent converges to a stable control if the environment is stable (e.g., in the context of Pensieve, when the network path has a fixed available bandwidth). At the same time, it is reasonable to assume that the RL agent must adapt in response to changes in environment. A key challenge is that neither of these concerns can be guaranteed pre-hoc; it is unclear how a trained RL agent will respond to conditions that are not seen during training. The only option today, it appears to us, is to keep an eye on how the RL agent behaves in practice. Systematically checking that the agent's decision achieves these desiderata is a worthy direction for future work.

Local maxima policies. RL algorithms try to find the values of parameters that will maximize their reward function. Because it is a non-convex problem, it is possible that the network will converge to a local maximum instead (as shown in Section 4). This observation leads to three major challenges in analyzing the model:
(1) Detect when the learnt policy is sub-optimal.
(2) Guide training towards improvement.
(3) Identify, during training, if the model appears to be stuck in a local maximum.

8 CONCLUSION

We have shown that by observing the decisions of agents and using simple, easily available analysis tools such as LIME, it is possible to extract some information about the behavior of the Pensieve agent. This method allows us to look into the "black-box" models and, as we show here, can offer insights that are useful to both model creators as well as users of the model. More work needs to be done however and we discuss some additional issues specific to understanding RL models. Ensuring the correctness of ML solutions remains an interesting and open problem that needs to be solved to encourage the deployment of RL-based systems in production environments.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their feedback. We are grateful to Nikolaj Bjørner, Bernard Ghanem, Hao Wang and Xiaojin Zhu for their valuable comments and suggestions. We also thank the Pensieve authors, in particular Mohammad Alizadeh and Hongzi Mao, for their help and feedback.

REFERENCES

[1] A. Dethise, M. Canini, and S. Kandula. Cracking Open the Black Box Github repository, 2019. https://github.com/adethise/observing-rl-agents-netai2019.
[2] J. Chen, L. Song, M. Wainwright, and M. Jordan. Learning to Explain: An Information-Theoretic Perspective on Model Interpretation. In *ICML*, 2018.
[3] M. Dong, T. Meng, D. Zarchy, E. Arslan, Y. Gilad, B. Godfrey, and M. Schapira. PCC Vivace: Online-Learning Congestion Control. In *NSDI*, 2018.
[4] N. Feamster and J. Rexford. Why (and How) Networks Should Run Themselves. *CoRR*, abs/1710.11583, 2017.
[5] FICO. Explainable Machine Learning Challenge, 2018. https://community.fico.com/s/explainable-machine-learning-challenge.
[6] H. Mao, R. Netravali and M. Alizadeh. Pensieve Github repository, 2017. https://github.com/hongzimao/pensieve.
[7] K. Kompella. The Self-Driving Network™: How to Realize It, 2017. https://www.nanog.org/sites/default/files/1_Kompella_The_Networking_Grand_Challenge.pdf.
[8] S. M. Lundberg and S.-I. Lee. A Unified Approach to Interpreting Model Predictions. In *NIPS*, 2017.
[9] M. T. Ribeiro, S. Singh and C. Guestrin. LIME Github repository, 2016. https://github.com/marcotcr/lime.
[10] H. Mao, M. Alizadeh, I. Menache, and S. Kandula. Resource Management with Deep Reinforcement Learning. In *HotNets*, 2016.
[11] H. Mao, R. Netravali, and M. Alizadeh. Neural Adaptive Video Streaming with Pensieve. In *SIGCOMM*, 2017.
[12] 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*, 518(7540):529, 2015.
[13] NIPS. Interpretable ML Symposium, 2017. http://interpretable.ml/.
[14] M. T. Ribeiro, S. Singh, and C. Guestrin. "Why Should I Trust You?" Explaining the Predictions of Any Classifier. In *KDD*, 2016.
[15] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *ICLR workshop*, 2014.
[16] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar. Learning to Route. In *HotNets*, 2017.
[17] A. Verma, V. Murali, R. Singh, P. Kohli, and S. Chaudhuri. Programmatically Interpretable Reinforcement Learning. In *ICML*, 2018.
[18] C. Wierzynski. The Challenges and Opportunities of Explainable AI, 2018. https://ai.intel.com/the-challenges-and-opportunities-of-explainable-ai/.
[19] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In *SIGCOMM*, 2015.

A REPRODUCIBILITY APPENDIX

Our repository [1] provides the source code used for the experiments in this paper. We also include a copy of the trained agent and the network traces provided by the Pensieve authors [6] used in our experiments.

Our code includes scripts to convert the network traces to execution traces, generate explanations on those traces using LIME, and finally create the plots included in the paper. The experiments and data generation can be run on any machine capable of running Python3 and TensorFlow.

We do not include the training information for the agent being tested. To train a new agent and perform its analysis, we refer the reader to the original Pensieve paper.

This section describes the requirements to re-run our experiments, how data is generated and transformed, and finally how data is aggregated for visualization.

A.1 Requirements

Platform. Reproducing our results does not require any specific hardware. The following software can be used to reproduce our experiments and measurements:

- `Python` 3.7.3
- `TensorFlow` 1.13.1
- The following Python modules, available from pip, are needed: `tflearn`, `numpy`, `lime`
- To display the results, `Jupyter` and `matplotlib` are required

Data. We require the user to supply the following data:

- The checkpoint of a trained agent based on Pensieve
- Traces of network bandwidth measurements. The format is `<timestamp [s]> <bandwidth [MB/s]>`
- The per-chunk size (in bytes) of the video for each available bit rate

Our implementation is immediately ready to run with any agent sharing the interface of Pensieve, including new agents trained with different data or parameters. Agents sharing the Pensieve interface can be easily adapted by implementing the `Predictor` interface, which is independent of the underlying ML framework.

Dataset. Our experiments use the trained Pensieve agent provided by the authors [6], one video divided in 48 chunks and 6 bit rates, and 142 network traces. In total, we analyze 6816 decisions.

A.2 Data generation

Once a new agent and fresh network data have been collected, reproducing the results requires running the `Predictor`. This will convert the network traces to execution traces, which store the environment values, outputs, and rewards at each decision step for all traces.

For this conversion, we provide two scripts that will automatically run the agent on all traces. One script (`run_agent.py`) runs the normal agent without modification. The second script (`run_agent_top_k.py`) takes as a parameter some value k and will run the agent by including only k of its features, which can be chosen by ordering all feature by order of importance. Note that the importance of features will be determined later using the traces of the normal

agent, but for convenience our results already include all features by order.

After this step, we acquire as many execution traces as we have network traces. Each trace contains all environment parameters and decisions. We use this data to train a *LIME tabular explainer*. This explainer takes as input the input values for the neural network, and returns the contribution of all features for each class (which can be positive or negative). The explainer is otherwise set to use default parameters. For details about LIME and LIME explanations, we refer the reader to their repository [9]. We do not generate explanations for the traces using the top k features, as those are not required for this paper.

To extract information from the explanations more easily, we implemented our own explanation model. It is suitable for sequences of decisions, as each trace's explanations can be queried by the user for the values and weights of all or any feature over time.

A.3 Figure data

We describe how the data is aggregated for all figures in the paper.
Bandwidth: *(Figure 2)* The bandwidth measurement is the average available network bandwidth for each trace. This information is encoded in the trace files. We collect this network speed over the whole trace, which might extend beyond the time used for video streaming. We have 142 bandwidth values.
Bit rate probabilities: *(Figures 3, 4)* For each decision taken by the agent, we collect a vector with the probabilities to select each bit rate. Those vectors are then summed and normalized to reflect the global probabilities. We have 6816 vectors of 6 bit rate probabilities.
Throughput: *(Figures 4, 5, 11, 13, 16)* The throughput is the value of goodput achieved by the system for downloading each chunk. It is also used as one of the inputs to the Pensieve agent. We have 6816 throughput values.
Expected bit rate: *(Figures 5, 6, 11, 15, 16, 17)* The expected bit rate for one decision is the average of the probability vector weighted by the bit rate values.
Reward: *(Figure 7)* The reward for each decision is computed from Formula 1. It is expressed in Mbps with $\mu = 4.3$. We have 6816 reward values.
QoE: *(Figure 10)* The QoE for one trace is the sum of rewards.
Feature weight: *(Figures 9, 12, 13, 14)* For each decision and each feature, LIME returns how much the feature contributed to the decision (which can be a negative value). We take the sum of the absolute values for each bit rate as our feature weight. We then normalize all weights so that the total of all features is equal to 1. We have 6816 weights for each of the 25 features.
Data in buffer, previous bit rate *(Figures 12, 14, 15, 17)* These values are collected from the Pensieve simulator and are also used as inputs to the Pensieve agent for each decision.