Article

World-Models for Bitrate Streaming

Harrison Brown [1], Kai Fricke [1] and Eiko Yoneki [1,*]

[1] University of Cambridge
* Correspondence: eiko.yoneki@cl.cam.ac.uk

Abstract: Adaptive bitrate (ABR) algorithms optimize the quality of streaming experiences for users in client-side video players especially in unreliable or slow mobile networks. Several rule-based heuristic algorithms can achieve stable performance, but they sometimes fail to adapt properly to changing network conditions. Fluctuating bandwidth may cause algorithms to default to behavior that creates a negative experience for the user. ABR algorithms can be generated with reinforcement learning, a decision-making paradigm in which an agent learns to make optimal choices through interactions with an environment. Training reinforcement learning algorithms for bitrate streaming requires building a simulator for an agent to experience interactions quickly; training an agent in the real environment is infeasible due to the long step times in real environments. This project explores using supervised learning to construct a world-model, or a learned simulator, from recorded interactions. A reinforcement learning agent trained inside of the learned model, rather than a simulator, can outperform rule-based heuristics. Furthermore, agents trained inside the learned world-model can outperform model-free agents in low sample regimes. This work highlights the potential for world-models to quickly learn simulators, and to be used to generate optimal policies.

Keywords: Reinforcement learning; bitrate streaming; world-models; video streaming; model-based reinforcement learning

1. Introduction

Determining bitrates for a positive user experience in unreliable or slow mobile networks can be difficult. Hardcoded algorithms may fail to accurately predict future bandwith, leading to rebuffering, and users may prefer minimal levels of quality fluctuations. An unreliable network may cause algorithms to select the lowest quality by default; selecting the highest level that bandwidth can support may not satisfy user preferences. Existing computer systems tackling this and other related problems typically make decisions based on heuristics in the form of configuration parameters, or human-engineered algorithms. However, humans may lack the intuition to tune parameters or design decision-making algorithms. Reinforcement learning is a machine learning paradigm where an agent can learn to make optimal decisions through interactions with an environment. The reinforcement learning paradigm has been applied successfully to bitrate streaming in the Pensieve project, outperforming rule-based approaches [1]. However, many reinforcement learning algorithms have high sample-complexity, requiring tens of millions of interactions with the real environment to learn an optimal policy. Training these algorithms can be severely constrained by the run time and lack of parallelization in real systems environments. Human-engineered simulators, such as a simulator for bitrate streaming, can replace real environments for training. However, building simulators may require significant developer effort to represent the complexities of the environment. Typically, the reinforcement learning algorithms applied in practice are model-free; while these algorithms learn a policy to make optimal decisions, they do not learn the dynamics of the environment [2]. Conversely, model-based algorithms learn a model of how the environment will behave, and then use the learned model to derive a policy [3]. World-models are a type of model-based reinforcement learning where a learned model can simulate all aspects of the environment [4]. Recent work [5,6] has shown it is possible to train agents entirely inside learned world-models of complex environments, rather than

the real environments themselves. Successfully learning world-models could provide vast benefits: cheap parallel training, faster training times, safe experimentation, and increased performance through planning. This work explores whether it is possible to build world-models for bitrate streaming, and whether world-models can can provide any benefits in learning optimal policies. We provide experiments that show the performance of an agent trained entirely inside a learned world-model, and provide sample complexity comparisons with current model-free algorithms. It demonstrates that world-models can successfully learn simulators from recorded data and world-models can outperform model-free approaches in low sample regimes.

2. Background

2.1. Bitrate Streaming

Streaming video over variable-bandwidth networks requires a client to adapt to network conditions to create the optimal user experience [7]. Users like to have high video quality, minimal buffer pauses, and no noticeable differences in quality between video chunks; these preferences vary between users and situations. HTTP-based adaptive streaming, through standards such as DASH [8], is the predominant form of network streaming today. In DASH, videos are stored on servers in multiple chunks, each holding a few seconds of video. A client-side application makes requests to download chunks, allowing servers to have stateless backends. Each chunk is encoded at several discrete bitrates, with higher bitrates containing higher resolution and chunk sizes. Quality-of-Experience (QoE) is the primary metric to evaluate the performance of the client. A variety of QoE metrics exist, and many metrics are a function of three factors: maximal resolution, buffering time, and smoothness (rapid changing of bitrates across consecutive chunks) [1]. Selecting the optimal bitrate is challenging due to the variability of network conditions [9], and the conflicting QoE requirements. Moreover, bitrate decisions can have a cascading effect on QoE over the course of the video; for example, a high bitrate may drain the playback buffer, causing the video to rebuffer in the future [1]. The QoE metric is defined below for a video of N chunks:

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

Here, R_n is the chosen bitrate, $q(R_n)$ is a quality function on the chosen bitrate, μ is a parameter constant, T_n is the time to download the chunk, and the final term is the penalty for consecutive changes in resolution. There are various heuristic-based algorithms for bitrate control. RobustMPC [10], for example, decides the next bitrate by maximizing QoE across predicted future bitrates in a near horizon, computed through a harmonic mean; however, it can conservatively estimate throughput and requires tuning. Heuristics, such as MPC, may perform extremely poorly when their network assumptions are violated [1]. Pensieve [1] was the first formulation of bitrate streaming as a reinforcement learning problem, and achieved state-of-the-art results on the problem, outperforming human-engineered heuristics, including RobustMPC [10], BOLA [11], and buffer-based [12] algorithms.

2.2. Reinforcement Learning

Reinforcement learning is a decision making paradigm. In reinforcement learning, an agent takes actions in an environment to optimize a cumulative reward. Reinforcement learning problems can be stated as a Markov Decision Process, formally defined with a 4-tuple:

- $\mathcal{S}$ is the set of possible states
- $\mathcal{A}$ is the set of possible actions available to an agent
- $\mathcal{P}_{s_t,a,s_{t+1}}$ is the transition kernel for states and actions, the probability that an action a at state s leads to state s'
- $\mathcal{R}_{s_t,a,s_{t+1}}$ is the reward function, the immediate reward from transitioning to state s' from s due to action a

A policy π, produces an action a given current state s: $\pi(s) \rightarrow a$. The objective of a reinforcement learning algorithm is to learn a policy that maximizes the expected return in an environment.

$$\pi \quad \mathbf{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}))]$$

Here, $R(s_t, a_t, s_{t+1})$ is the reward given by the policy π selecting action a at time t, and $\gamma \in [0, 1]$ is the discount factor that determines the importance of future rewards. For $\gamma = 1$, if the environment does not contain a terminal state, or the agent never reaches one, rewards can become infinite. Typically, reinforcement learning algorithms are evaluated in their performance across episodes, sequences of actions that ends with a terminal state. Agents initially do not have any knowledge of environments; agents must balance between exploration of unseen states and exploitation of current knowledge to find an optimal policy.

Partially Observable Markov Decision Processes (POMDPs) are MDPs where the state space may be noisy or not fully observable. Reinforcement learning is concerned with learning a policy when the environment model is not known in advance. There are two broad classes of reinforcement learning algorithms: model-free and model-based [3]. Generally, model-free methods learn a policy without learning a model of environment dynamics, and model-based methods learn a model of environment dynamics and use the learned model to derive a controller. In practice, there are no distinct boundaries between these two classes, and some algorithms blur the lines between the categories [13]. Classical reinforcement learning methods can be untenable on large state and action spaces; the state space may be too large for the agent to visit exhaustively, and there are large memory overheads for dynamic programming methods. Deep reinforcement learning has been used to achieve state-of-the-art results on tasks that were previously computationally prohibitive, such as Atari games [14] and Go, a complex board game [15]. Agents trained with deep reinforcement learning can learn complex functions, handle unstructured environment data, and predict quality actions in states that have never been visited. There are a wide variety of reinforcement learning algorithms with different use cases and advantages, such as parallelization or properly handling discrete action spaces. Actor-critic methods [16] have achieved recent success on Atari games, StarCraft [17], and visual navigation [18] among others. In actor-critic algorithms, a critic network estimates a value-function, and an actor network updates a policy-network based on the estimations from the critic [19]. This work uses the A2C algorithm, a synchronous version of the A3C algorithm. For brevity, this work omits the mathematical details of the A2C algorithm, and refers the reader to the original A3C paper [16] for more information.

2.3. Motivation for World-Models

Model-free reinforcement learning is notoriously sample inefficient; agents may require tens or hundreds of millions of interactions to learn complex tasks. Rainbow, a combination of six extensions to the DQN algorithm, achieved state-of-the-art results for sample efficiency, but still required 44 million frames to beat existing baselines in the Arcade Learning Environment [20]. Most current reinforcement learning research is evaluated on simulators, games in the Arcade Learning Environment, or other video games. While these control problems may be complex and challenging, one interaction with the environment is typically sub-second. Conversely, many real-world systems can take several seconds to execute one step. Custom-built simulators are often necessary to successfully train reinforcement learning agents in these settings. For example, a simulator for bit-rate streaming can allow an A3C-based agent to experience 100 hours of video in only 10 minutes of training time [1]. In contrast, training the same agent on the real system would take more than 10 years to receive enough experience to reach comparable performance [7]. Building simulators may be difficult; custom simulators require developer effort and domain knowledge, and built simulators may not accurately

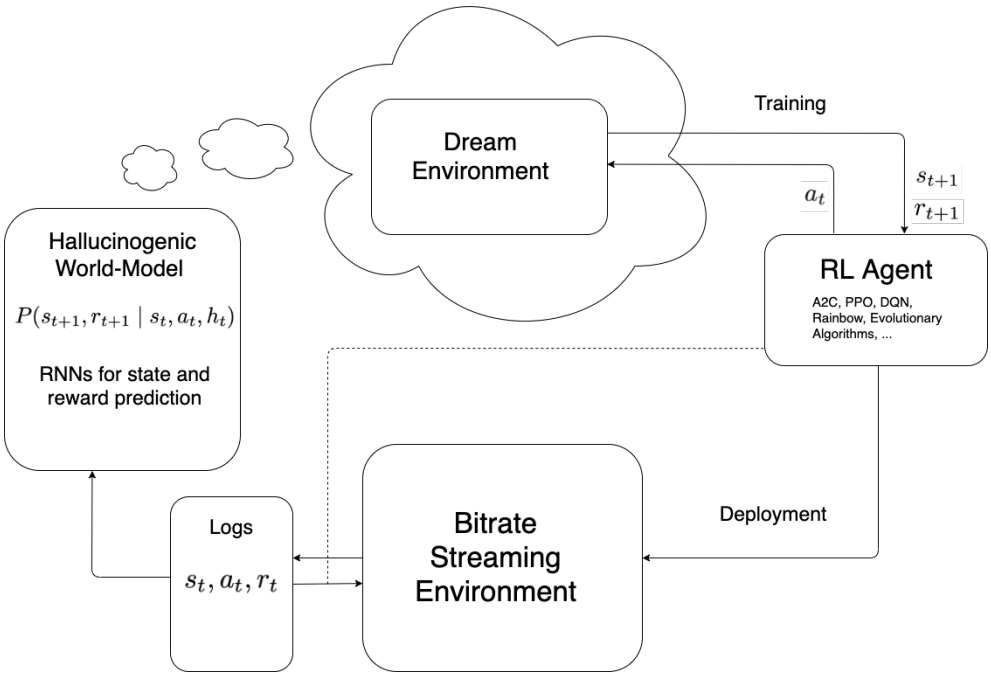

Figure 1. World-model pipeline. An agent trains inside the learned world-model and then is deployed back into the real environment.

model real-world systems [21]. Some reinforcement learning methods that require high sample complexity, such as policy gradient methods, cannot be used if the real environment's interactions take too long or the simulator is too slow [22]. If one can learn a world-model from recorded data, one could train an agent inside the world-model without ever interacting with the real system. Agents can train cheaply in parallel with learned models. A learned environment can execute steps without waiting in real-time for real-world events, rendering states, or completing computations unrelated to essential aspects of the environment.

3. World-Models

Model-based RL encompasses a wide variety of methods, including world-models. In this project, world-models refer to models of the environment that can be fully substituted in place of the real environment. Some instances of model-based learning may learn the components for a full world-model, but may not use the model to replace the real environment for training. World-models typically involve two components, the model of the environment, M, and a controller, C [4]. A model learns from real interactions with an environment, and these interactions must be gathered from a controller or logs of previous interactions. A world-model must learn to predict the next state s_{t+1} and reward r_{t+1}, given the current state s and action a. Once a world-model has learned $P(s_{t+1}, r_{t+1} \mid s, a)$, M can be substituted in place of the real environment as a world-model. Figure 1 shows a visualization of a world-model pipeline. First, a world-model is learned from recorded observations. A controller then learns inside the world-model's hallucinated episodes of the real environment, which can be transferred back to the real environment during deployment. A controller may learn policies fully inside of the world-model [6], use the world-model for planning [23], or use a world-model to augment the real environment [5]. Neural networks can be used as function approximators to model the environment. In deterministic, predictable environments, feedforward neural networks can approximate the world-model. However, these networks tend to fail in POMDP problems [24]. Memories of previous events can assist with the difficulties of POMDPs. The Recurrent

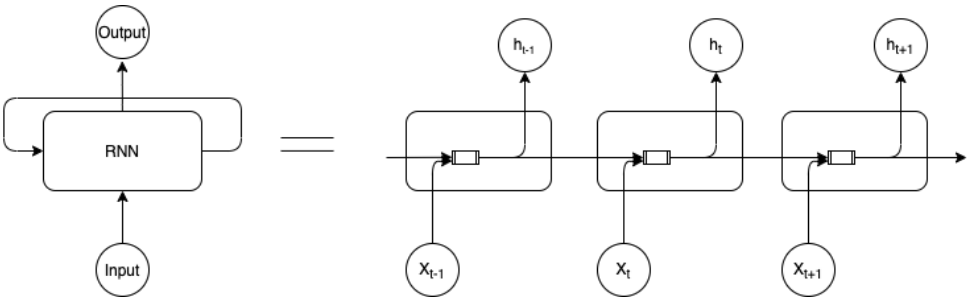

Figure 2. Recurrent Neural Networks allow the network to retain information over time. Figure adapted from [25].

Neural Network (RNN) architecture is a class of neural network models that process temporal data. The following subsection provides an overview of the RNN architecture.

3.1. Generative Models

RNNs can recognize patterns with dynamic temporal behavior and thus serve as the foundation for many world-models. At each step t, a simple recurrent neural network accepts an input x_t, and outputs a value h_t. The RNN passes information from one step of the network to the next. The left side of Figure 2 displays the recurrent nature of an RNN, and the right displays an unfolded version of a simple RNN. More complex architectures are used in practice; Hochreiter et al. [26] introduced Long Short-Term Memory (LSTM), a variant of the RNN designed to handle long term dependencies in input sequences.

The output of an RNN is deterministic; however, an RNN can output the parameters for probabilistic model to introduce stochasticity [25]. The output distribution is dependent on the hidden state in the RNN, a representation of previous inputs. Mixture Density Networks (MDNs) [27] are a class of models that output the parameters to a Gaussian Mixture Model (GMM), a probabilistic model composed of several Gaussian distributions with different weights. GMMs contain three parameters: μ, the centers of each component, σ the scales of each component, and π, the weights of each component. Rather than performing a regression, this model maximizes the likelihood that a model gives rise to a set of data points. This model can, in theory, approximate any arbitrary continuous probability distribution. Graves [25] combined a Mixture Density Network with an RNN (MDN-RNN), and used the network to achieve state-of-the-art results on handwriting generation. In a prediction network, the hidden state of the model passes through another layer, outputting the parameters to a mixture model y_t, a distribution modeling the input at the next time step. During inference, a sample from the mixture model, $P(x_t \mid y_{t-1})$ is then fed as the next input into the network. This structure allows a RNN to generate entire sequences, where each output is conditioned on a representation of past states.

3.2. MDN-RNN World-Model

Ha & Schmidhuber [5] used the MDN-RNN to create a hallucinogenic world-model in reinforcement learning environments, achieving state-of-the-art results on the Car Racing and VizDoom tasks. Their work is the primary inspiration for the work in this paper. The Car Racing and VizDoom tasks are stochastic, and the bitrate streaming environment is partially determined by a stochastic input process, the network bandwidth. The MDN-RNN can handle random discrete events, such as a monster shooting a fireball in VizDoom. While sampling in MDN-RNN is probabilistic, it can still form approximations for deterministic properties. The MDN-RNN can fit naturally to bitrate streaming, as bitrate streaming contains discrete events, such as bandwidth dropping while passing through a tunnel.

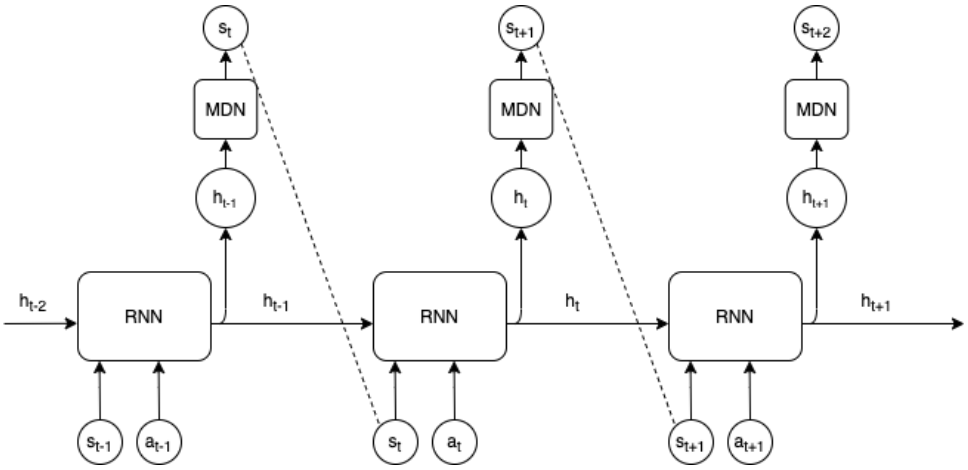

Figure 3. A state predicting MDN-RNN. Figure adapted from [5].

Ha & Schmidhuber [5] first used a variational autoencoder to compress the state space s into a latent vector z, and learned a world model with this compressed version of the state space. The authors train an MDN-RNN to model $P(z_{t+1} \mid z_t, a_t, h_t)$, where a_t is the action take at time t, z_t is the state of the environment encoded as a latent vector at time t, and h_t is the hidden state of the MDN-RNN at time t. The world model is trained using recorded episodes in the teacher forcing method, where the inputs to the MDN-RNN model in training consist only of the ground truth at each step. Sampling only occurs during the training of the controller. For inference, the MDN-RNN uses a softmax layer on the output π's to form a categorical distribution. The softmax function, $\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$, converts a vector of numbers into a probability distribution where each component is in the interval (0,1), and the entire vector sums to 1. A sample drawn from this distribution determines which Gaussian distribution pair, (μ_i, σ_i), is used to sample the actual output. Ha & Schmidhuber [5] additionally incorporate a temperature hyperparameter τ, which is used to adjust the randomness during sampling. The logits, components of the π vector before the application of the softmax function, are divided by the temperature parameter before being scaled into probabilities. τ is typically set between 0 and 1; however, larger values are used in practice. When τ trends to zero, the output of the MDN-RNN becomes deterministic, and samples only consist of the most likely point in the probability density function. Lower temperature values make sampling more conservative, and the unlikely candidates become even more unlikely to be sampled. Conversely, higher temperature values make unlikely candidates more likely to be sampled, resulting in more stochastic predictions.

During sampling, increasing the temperature parameter beyond one makes it possible to add more uncertainty and stochasticity into the dream environment. A controller trained in a hallucinogenic world-model with more stochasticity may be trained in a more difficult version of the real environment. Ha & Schmidhuber [5] found that increasing τ helps prevent a controller from taking advantage of the imperfections in the world-model, and can guide the controller to more conservative policies when deployed to the real environment. While the VizDoom and Car Racing environments have complex visual inputs, Ha & Schmidhuber did not model reward functions in their experiments. For the hallucinogenic VizDoom world-model, the reward is simply incremented by a constant, 1, at each time step. Most systems problems do not contain reward functions that are this trivial.

Ha & Schmidhuber [5] combined an incremental reward function and MDN-RNN state predictor to form a fully hallucinogenic world model. A controller receives current state s, and takes an action a.

These are concatenated together, and the RNN outputs the parameters to the MDN. The world-model presents a sample taken from the MDN as the next state to agent, and the agent receives a reward if the state is not terminal. Figure 3 displays a visualization of how states are sampled over time in the world-model.

4. Design

It is straightforward to formulate bitrate streaming as a reinforcement learning problem. The state s is a vector $(x_t, \tau_t, n_t, b_t, c_t, l_t)$, where x_t is the throughput measurement, τ_t is the download time of the past chunk, n_t is a vector of the m available sizes for the next chunk, b_t is the size of the buffer, c_t is the number of chunks remaining in the video, and l_t is the selected bitrate of the previous chunk. The action a is the bitrate for the next chunk, and the reward r per time step is the contribution to the QoE function for time step t.

Many reinforcement learning algorithms are evaluated on collections of environments, such as the Arcade Learning Environment [28] , RLLab [29], and OpenAI Gym [30]. These collections have a common interface and allow researchers to experiment with novel algorithms easily. Moreover, they allow researchers to easily reproduce and evaluate results. Until recently, there was no easy-to-use platform for experimenting and benchmarking problems in the computer systems domain. Many applications of reinforcement learning in computer systems require difficult configurations and the collection of traces in production systems. Park [7] is a attempt to lower the barrier to entry to machine learning research in the systems domain. The Park platform consists of 12 systems-centric environments, including both custom-built simulators and interfaces to real system environments. Park includes a bitrate streaming simulator that is based on the work in the Pensieve project [1]. This project leverages the Park simulators for ease of development and training. Algorithms to learn hallucinogenic world-models must successfully work on these simulated systems environment before they can be applied to larger, more complex systems, such as an entire data center. Park environments use an interface similar to OpenAI Gym; agents provide a valid action to the environment and receive a tuple containing the next state, the reward, and a boolean value marking the end of the episode. Multiple instances of the same Park environment can easily run concurrently, generating experience in parallel to speed up training and evaluation. We ported the Park environment to OpenAI Gym for easier installation and modification.

The default Park formulation uses the linear QoE, a linear combination of the bitrate and stall, for its quality function. The Park environment [7] supports a set of network traces, containing a 3G/HSDPA mobile data set collected in Norway from commuters in transit [31], such as bus or train passengers. In order to avoid situations where bitrate selection is trivial (consistent high network throughput) or impossible (little to no service), the data set only contains traces where the average throughput is less than 6 Megabits per second (Mbps), and the minimum throughput is greater than 0.2 Mbps [1]. In addition, we have added a test set of extended traces from the Pensieve project to the Park environment.

4.1. World-Model Design

A fully hallucinogenic world-model requires an accurate state and reward predictor. It is straightforward to apply the MDN-RNN to predict state in the bitrate streaming environment. The state space for bitrate streaming is continuous, except for the number of chunks remaining in the video, which decreases by 1 until the episode ends. These experiments apply the MDN-RNN to the entire state space and assume that the remaining chunk predictions will contain a small rounding error.

The reward for bitrate streaming, QoE metrics, measures the quality of experience for the user across the episode but is computed at each step. This computation involves the previous state

and action, and the current state and action. The action prior to each step, a_{t-1}, is recorded in the state space at each step, s_t. If an MDN-RNN is used to predict the next state given an action, we can apply a function to compute the contribution to total QoE at each step, forming a fully hallucinogenic world-model. This method also has the advantage of allowing the QoE metric for performance to be substituted or changed as needed, without retraining the entire model. The reward function at each step is as follows:

$$\text{Reward}_{t+1} = q(a_t) - \mu(\max(\tau_t - b_{t-1}, 0)) - \mid q(a_t) - q(a_{t-1}) \mid$$

Here, μ is the linear QoE parameter constant, (4.3 in the simulator), $q(a_t)$ is the quality function on the current action, $\max(\tau_t - b_{t-1}, 0)$ is the rebuffering time, computed from the previous buffer size and the time to download the current chunk. $\mid q(a_t) - q(a_{t-1}) \mid$ is the penalty term for bit rate change from the previous action. The world-model presents a state to the agent s_t, and the agent selects an action a_t. The MDN-RNN samples the next state s_{t+1}, and the world-model then computes the reward using the three inputs it requires: $\text{Reward}_{t+1}(s_t, a_t, s_{t+1})$. The world-model then passes the next state and computed reward to the controller. In the first step of the environment, the bitrate change penalty is set to 0 because the agent does not have a previous action.

Ha & Schmidhuber [5] used evolutionary algorithms to train their controller inside the world-model. Instead, we evaluate the ability of a model-free reinforcement algorithm to train inside the world-model.

4.2. Training Methodology

In theory, model-based algorithms can provide benefits to many systems problems; however, world-models has been seldom explored on non-robotic real-world problems. While metrics such as mean-squared error or log-likelihood give an indication of performance and are used to train algorithms, these measures do not provide an intuitive measure of the robustness of a world-model. This work considers a world-model to be sufficiently accurate if a model-free algorithm trained inside the hallucinogenic world-model can achieve similar performance to the same model-free algorithm trained in the real environment. A model-free agent trained inside of the hallucinogenic world model will likely not reach the performance of the same agent trained in the real environment, due to imperfections of the world-model.

First, this project establishes baselines for maximum performance of a model-free agent trained directly in the Park environment. Furthermore, we establish baselines for two heuristics in the Park environment, RobustMPC and Buffer-Based. The world-model described above can be constructed using recorded observations, and a model-free agent can train entirely inside the learned world-model. We adjust the number of interactions with the environment to establish sample complexity baselines, and make comparisons with a state-of-the-art model-free algorithm, Rainbow [20].

5. Results and Discussion

5.1. Baselines

For bitrate streaming, this work uses an implementation of the Advantage Actor-Critic (A2C) algorithm from Stable-Baselines [32], a collection of implementations of common deep reinforcement learning agents. We obtained a copy of the Park authors' bitrate streaming agent to validate the implemented agent used in this project. In correspondence, the authors noted that they added traces to the repository during publication and that results may have slight differences from the scores in the original paper. We performed a combination of grid search and refinement by hand on parameters for the value coefficient, starting entropy coefficient, entropy decay, and optimization algorithm. The

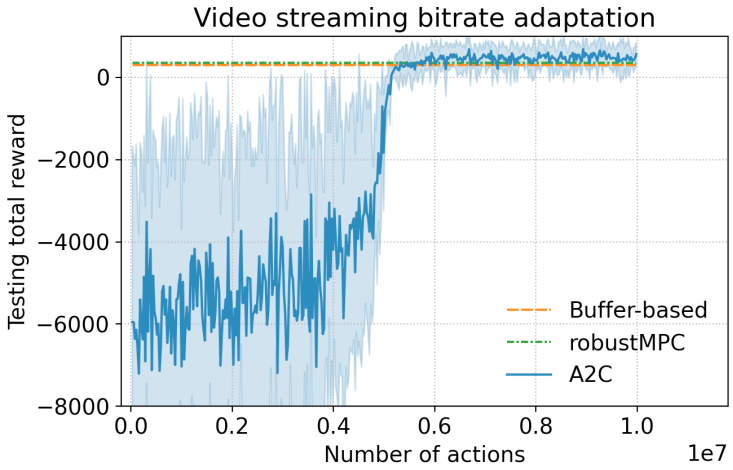

Figure 4. Training curve of an A2C agent in the bitrate streaming Park environment.

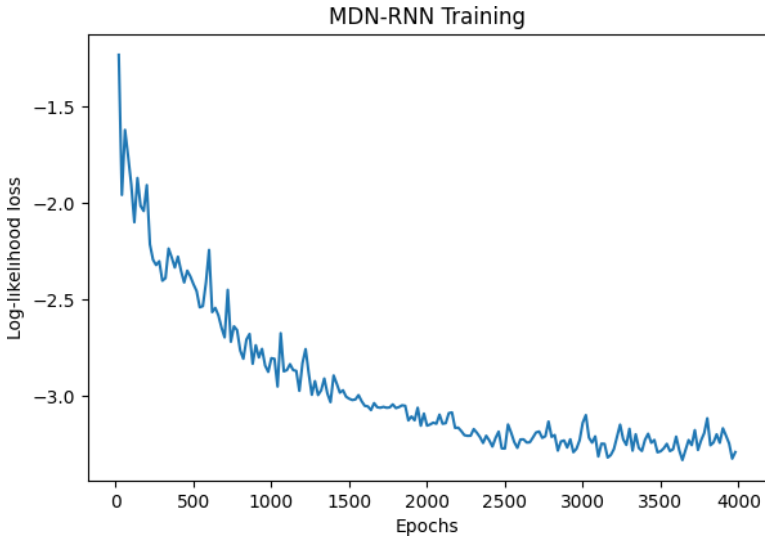

Figure 5. Training curve for an MDN-RNN with 64 units and 5 Gaussians.

entropy was decayed by the entropy rate every 264 steps and stops decaying once the entropy reaches zero. Figure 4 shows the training curve for the final model-free A2C agent trained entirely inside the Park environment.

Every 26,400 steps, the policy of the agent is evaluated for 10,000 actions. The solid blue line represents the average score at each evaluation step, and the shaded portion highlights the standard deviation. Because the environment is stochastic, the curve is jagged, as evaluating the agent's performance regularly with a high number of iterations impedes training. Through correspondence with the Park authors [7], we received their implementations for the buffer-based and RobustMPC heuristics. A random agent in the environment receives a score of -5729.3 ± 4250.1. The provided Buffer-based and RobustMPC agents receive average scores of 237.7 ± 435.4 and 286.2 ± 502.5 over 1000 episodes, respectively.

5.2. Training in a Dream

To construct the world-model, a script collects traces from 10,000 episodes (5 million total actions) episodes using a random agent; all observations are normalized using min-max scaling from the bounds of the Park environment. This project uses a version of the MDN-RNN from Ha & Schmidhuber

Temperature	*Dream Score*	*Real Score*	*Test Score*	*Real (Single Sweep)*
0.1	2064.1±1.7	−22010.7±694.1	−18215.6±	−22010.75±10986.6
0.5	1036.9±253.1	−3801.2±2115.6	−2304.6±	−4965.1±4222.8
0.75	649.5±136.8	−2461.0±2165.7	−1696.4±	−1159.8±1754.6
1	726.1±177.4	−437.6±776.2	−13.4±	−1317.7±1781.3
1.1	770.0±32.0	159.1±189.7	370.5±	61.2±810.6
1.2	569.7±242.2	220.3±92.9	339.4±	192.3±365.1
1.3	594.0±19.5	175.6±101.2	367.1±	290.1±367.2
1.4	406.8±34.0	235.6±120.4	370.6±	316.7±274.6
1.5	305.5±69.1	309.0±64.5	449.3±	315.3±349.0
1.75	−177.7±36.4	362.4±30.2	444.7±	396.3±279.9
2	−952.0±39.2	402.3±8.8	476.2±	392.2±193.5
2.25	−5686.5±5295.1	−1595.8±3052.8	−1178.3±	132.0±46.1
2.5	−7753.0±5860.0	−1585.1±3039.7	−1267.3±	202.6±383.1
3	−17087.6±5885.9	−3390.0±3106.3	−2570.8±	−5021.2±4105.5

Table 1. World-model temperature parameter sweep. The rightmost column contains the mean episode score from a single parameter sweep for each corresponding temperature. While a single sweep follows the general trend of increasing to a peak and then decreasing, the increase and subsequent decrease in scores are not always monotonic; the rightmost column highlights these discrepancies. Note the rightmost column contains the standard deviations of the episode scores for each agent, while the other columns contain the standard error of mean episode scores for three repeated runs.

[5], modified to operate in the native bitrate streaming state space, without an autoencoder. The MDN-RNN is wrapped by a class to advance the state of the world-model, and compute the reward function as described above. The wrapped world-model implements the OpenAI Gym interface; any controller compatible with this interface can train inside the hallucinogenic world-model. The following section contains the results from training both the MDN-RNN and training a controller inside the hallucinogenic world-model. The parameters used in these experiments are listed in Appendix A, and were the same parameters used by Ha & Schmidhuber [5], unless otherwise noted. Because the state space for bitrate streaming contains 12 values, this project trained the MDN-RNNs using a smaller hidden state size to reduce possibilities of overfitting. Figure 5 shows the log-likelihood loss during training for an MDN-RNN with 64 hidden units and 5 Gaussians. We trained the established baseline agent inside of the fully hallucinogenic world-model. We performed a sweep on the temperature values to determine how increased difficulty in the world-model provides benefits for translation to the real environment. Table 1 contains the results of the parameter sweep. For each temperature, an A2C controller was trained entirely in the world-model with that temperature. Next, the controller was evaluated across 500 episodes in the dream environment, the actual environment, and a test environment. This procedure was repeated three times for each temperature value. The first three columns in Table 1 shows the mean of the average score from each controller instance. At low temperatures, the agent can reach extremely high scores in the dream environment, but these scores do not translate to reality; an agent trained in a world-model with a temperature of 0.1 performs nearly four times worse than the performance of a random agent. As temperature increases, it

becomes progressively more difficult for the agent to earn high scores in the dream environment. Noticeably, the agents that translate the best to the real environment (see temperature of 2.0) receive poor overall scores in the dream environment. The temperature hyperparameter can be tuned to find the optimal translation from the dream environment to the real environment. The average across runs in Table 1 shows that increasing the temperature increases scores in the real environment, and then

Temperature	*Dream Score*	*Real Score*	*Test Score*	*Max Average Real Score*
1	−3532.04±2848.57	−1525.94±88.95	195.52±100.36	282.45±109.67
1.25	−5666.19±7003.71	−4158.25±2865.73	−1205.58±2329.62	132.80±38.21
1.5	−13128.57±10942.65	−437.6±6462.24	−3375.84±5514.53	320.92±113.37
2	−235.71±13.72	369.18±9.25	478.96±3.26	378.79±309.28
2.25	−1628.17±7.91	429.08±11.65	557.84±5.87	441.09±316.88

Table 2. Temperature parameter sweep for a world-model trained from 1M recorded actions. The rightmost column contains the maximum mean episode score from an agent trained inside the world-model at the corresponding temperature. Note the rightmost column contains the standard deviations of the episode scores for that agent, while the other columns contain the standard error of mean episode scores for three repeated runs.

decreases scores as the environment becomes too difficult to find a translatable policy. However, this behavior is not entirely consistent if one were to run a single sweep on temperatures. The rightmost column in Table 1 shows the average scores and standard deviations a single temperature sweep, tested in the actual environment. While a single sweep follows the general trend of increasing to a peak and then decreasing, the increase and subsequent decrease in scores are not always monotonic; the rightmost column in Table 1 highlights these discrepancies. In training, it is possible that an agent learns to exploit the world-model and fails to generalize to the real environment. The best A2C agent trained in the world-model received an average score of 408.71 ± 280.6 in the real environment, which is close but does not beat the average score of 500.5 ± 358.8 for the same model-free agent. The world-model was trained with a total of 5 million actions in the real environment, and the controller trained for 10 million actions inside of the world-model. The strictly model-free controller used 10 million actions in the real environment. While the model-free controller took more samples in the real environment for these experiments, these experiments do not show conclusive benefits concerning sample complexity.

5.3. Sample Complexity Bounds

5.3.1. Rainbow

To make fair comparisons on sample complexity, we compare our results with the Rainbow algorithm [20]. Rainbow is considered to be state-of-the-art with regard to sample complexity, and was used recently in comparisons with a model-based algorithm, SimPLe [6]. We used a Rainbow implementation from the Dopamine package; further information is provided in Appendix B. We reduced the number of actions in the real environment to 1,000,000 (1M) for world-model construction and model-free training. For hyperparameter tuning, we used a combination of grid search and autotuning to optimize Rainbow. The grid search uses a nearly identical parameter space to [6]; these parameters are listed in Appendix B. We additionally did 100 iterations of hyperparameter autotuning with the Optuna framework [33] to optimize the hyperparameter search space with the Tree-Structured Parzen Estimator method (TPE). TPE attempts to balance exploration versus exploitation by iteratively selecting hyperparameters with the largest expected improvement on model loss. For more information on TPE, this work refers readers to the original paper [34]. The best Rainbow model trained for 1M actions received an average score of 306.35 ± 316.34 in an evaluation of 500k actions.

5.3.2. World-Model Comparisons

We performed experiments training a model-free agent inside of a world-model trained from lower sample complexities: 1M and 500k actions. Table 2 and Table 3 show the results from these experiments. Training with lower sample complexities results in much less stability, but still allows

Temperature	*Dream Score*	*Real Score*	*Test Score*	*Max Average Real Score*
0.75	550.04±22.86	−813.62±764.66	−403.87±659.83	68.85±826.89
1	453.31±12.76	96.09±50.03	382.93±30.09	151.88±826.91
1.25	291.33±61.87	−37.10±132.88	247.92±87.29	82.31±648.22
1.5	158.27±126.67	280.53±182.38	428.88±83.72	399.79±245.12

Table 3. Temperature parameter sweep for a world-model trained from 500,000 recorded actions. The data follow the same format as Table 2.

Num. Actions	Agent	Mean Episode Score	STDEV
-	Random Agent	-5729.3	- 4250.1
-	Buffer-based Agent	237.7	435.4
-	RobustMPC Agent	286.2	502.5
5M	A2C Model-free Agent	500.5	358.8
5M	Best World-model Agent	408.71	280.5
1M	Best World-model Agent	441.0	316.8
1M	Best Rainbow Agent	306.3	316.3
500k	Best World-model Agent	399.7	245.1

Table 4. Comparison between heuristics, model-free, and world-model agents for bitrate streaming.

for performance that is nearly identical to the 5M-sample world-model. During some runs, an agent would learn policies that translate poorly into the real environment. Higher temperatures for the 1M-sample world-model in Table 2 result in greater world-model stability, and the performance was very similar between each repeated run.

The maximum average score of an agent trained in a world-model with only 500k samples achieved a score of 399.79 ± 245.12, which is higher than the average score of the best Rainbow agent trained with 1M samples. There is no reason to believe that the Rainbow agent would improve performance with a lower sample complexity, and thus we did not do further training for this sample regime.

5.4. Summary

An agent trained entirely inside a world-model was able to achieve performance that was close to strictly model-free agent. Furthermore, an agent trained inside the world-model beats the scores of the provided heuristics. Additionally, these world-model agents learn more consistent policies that have lower variance when compared to the heuristics. The best world-model agents do consistently learn policies with lower variance than the model-free agents. Finally, the world-model agent was able to outperform a well-tuned Rainbow agent for lower sample complexities.

5.5. Future Work

Ha & Schmidhuber [5] passed the hidden state of the world-model to their evolution-based controller at each step. Further work may incorporate passing the hidden state of the world-model to a model-free controller.

Interpretability has become an issue across all areas of machine learning [35]; a human cannot explain decisions made by many machine learning algorithms. While not all systems require interpretability, it is important to have solutions that are easy to understand and debug. Heuristics are often easy to understand, and can be easily engineered with specific safety properties [7]. When learning world-models on real systems, it may be essential to have a world-model detect uncertainty or unexplored parts of the state space and provide this information to the controller. In these cases, a

controller trained in the world-model could fall back to a heuristic to avoid potentially damaging decisions.

6. Conclusions

This work does show that it can be straightforward to apply world-models to networking problems. The MDN-RNN world-model was able to capture the dynamics of the environment; however, these experiments required proper tuning of the temperature parameter to learn a policy that can translate into the real environment. Finding the optimal temperature hyperparameter for other environments may require sweeping several values, and these runs must be repeated. The optimal temperature was 70% higher than the optimal temperature used by Ha & Schmidhuber [5]. Furthermore, these experiments show a rapid decrease in performance after the sweep passes the optimal temperature. It may take careful tuning to find the optimal temperature in other environments. The experiments in this work demonstrate that world-models can successfully learn simulators from recorded data. Furthermore, world-models can outperform model-free approaches in low sample regimes. World-models are a promising area of research that may provide benefits for many other problems in the networking and systems domains.

Acknowledgments: This research was supported by the Alan Turing Institute.

Abbreviations

The following abbreviations are used in this manuscript:

ALE	Arcade Learning Environment
A2C	Advantage Actor-Critic
A3C	Asynchronous Advantage Actor-Critic
GMM	Gaussian Mixture Model
LSTM	Long Short-Term Memory
MDN-RNN	Mixture Density Network - Recurrent Neural Network
MDP	Markov Decision Process
MPC	Model Predictive Control
MSE	Mean Squared Error
Mbps	Megabits per second
POMDP	Partially Observable Markov Decision Process
RL	Reinforcement Learning
RNN	Recurrent Neural Network
TPE	Tree-structured Parzen Estimator
QoE	Quality-of-Experience

Appendix A Bitrate streaming MDN-RNN

Num Steps	4000
Input Sequence Length	12
Output Sequence Length	11
Batch Size	100
RNN Hidden State Size	0.00005
Number of Gaussians	5
Learning Rate	0.005
Grad Clip	1.0
Use Dropout	False
Learning Rate Decay	0

Table A1. Hyperparameters for bitrate streaming MDN-RNN

Appendix B Rainbow Hyperparameter Search

To make fair model-free comparisons in low sample regimes, we compare our results with the Rainbow algorithm. We tuned hyperparameters for a Rainbow agent from the Dopamine package: https://github.com/google/dopamine/blob/master/dopamine/agents/rainbow/rainbow_agent.py

Appendix B.1 Grid Search

60 iterations of grid search using the following hyperparameters. All runs used the default learning rate of 0.09.

- Target Update Period: $\{50, 100, 1000, 4000, 8000\}$
- Update Horizon: $\{1, 3\}$
- Update Period: $\{1, 4\}$
- Min Replay History: $\{500, 5000, 20000\}$

Appendix B.2 Autotuner

100 Iterations of Tree-Structured Parzen Estimator operating in following parameter space. The learning rate is a continuous interval; all other parameters ranges are integer intervals.

- Learning Rate: $(0.01, 0.15)$
- Target Update Period: $(50, 20000)$
- Update Horizon: $(1, 8)$
- Update Period: $(1, 8)$
- Min Replay History: $(100, 40000)$

References

1. Mao, H.; Netravali, R.; Alizadeh, M. Neural Adaptive Video Streaming with Pensieve. Proceedings of the Conference of the ACM Special Interest Group on Data Communication, 2017, pp. 197–210.
2. Arulkumaran, K.; Deisenroth, M.P.; Brundage, M.; Bharath, A.A. A Brief Survey of Deep Reinforcement Learning. *arXiv preprint arXiv:1708.05866* **2017**.
3. Kaelbling, L.P.; Littman, M.L.; Moore, A.W. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research* **1996**, *4*, 237–285.
4. Schmidhuber, J. On Learning to Think: Algorithmic Information Theory for Novel Combinations of Reinforcement Learning Controllers and Recurrent Neural World Models. *arXiv preprint arXiv:1511.09249* **2015**.
5. Ha, D.; Schmidhuber, J. Recurrent World Models Facilitate Policy Evolution. In *Advances in Neural Information Processing Systems 31*; 2018; pp. 2451–2463. https://worldmodels.github.io.
6. Kaiser, L.; Babaeizadeh, M.; Milos, P.; Osinski, B.; Campbell, R.H.; Czechowski, K.; Erhan, D.; Finn, C.; Kozakowski, P.; Levine, S.; others. Model-Based Reinforcement Learning for Atari. *arXiv preprint arXiv:1903.00374* **2019**.
7. Mao, H.; Negi, P.; Narayan, A.; Wang, H.; Yang, J.; Wang, H.; Marcus, R.; Shirkoohi, M.K.; He, S.; Nathan, V.; others. Park: An Open Platform for Learning-Augmented Computer Systems. Advances in Neural Information Processing Systems, 2019, pp. 2490–2502.
8. Akamai. dash.js. https://github.com/Dash-Industry-Forum/dash.js/, 2016.
9. Huang, T.Y.; Handigol, N.; Heller, B.; McKeown, N.; Johari, R. Confused, Timid, and Unstable: Picking a Video Streaming Rate is Hard. Proceedings of the 2012 Internet Measurement Conference, 2012, pp. 225–238.
10. Bemporad, A.; Morari, M. Robust Model Predictive Control: A Survey. In *Robustness in Identification and Control*; Springer, 1999; pp. 207–226.
11. Spiteri, K.; Urgaonkar, R.; Sitaraman, R.K. BOLA: Near-Optimal Bitrate Adaptation for Online Videos. IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications. IEEE, 2016, pp. 1–9.

12. Huang, T.Y.; Johari, R.; McKeown, N.; Trunnell, M.; Watson, M. A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. Proceedings of the 2014 ACM conference on SIGCOMM, 2014, pp. 187–198.
13. Chebotar, Y.; Hausman, K.; Zhang, M.; Sukhatme, G.; Schaal, S.; Levine, S. Combining Model-Based and Model-Free Updates for Trajectory-Centric Reinforcement Learning. Proceedings of the 34th International Conference on Machine Learning-Volume 70. JMLR. org, 2017, pp. 703–711.
14. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing Atari with Deep Reinforcement Learning. *arXiv preprint arXiv:1312.5602* **2013**.
15. Silver, D.; Huang, A.; Maddison, C.J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; others. Mastering the game of Go with deep neural networks and tree search. *Nature* **2016**, *529*, 484.
16. Mnih, V.; Badia, A.P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; Kavukcuoglu, K. Asynchronous Methods for Deep Reinforcement Learning. International Conference on Machine Learning, 2016, pp. 1928–1937.
17. Vinyals, O.; Ewalds, T.; Bartunov, S.; Georgiev, P.; Vezhnevets, A.S.; Yeo, M.; Makhzani, A.; Küttler, H.; Agapiou, J.; Schrittwieser, J.; others. StarCraft II: A New Challenge for Reinforcement Learning. *arXiv preprint arXiv:1708.04782* **2017**.
18. Zhu, Y.; Mottaghi, R.; Kolve, E.; Lim, J.J.; Gupta, A.; Fei-Fei, L.; Farhadi, A. Target-driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning. 2017 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2017, pp. 3357–3364.
19. Konda, V.R.; Tsitsiklis, J.N. Actor-Critic Algorithms. Advances in neural information processing systems, 2000, pp. 1008–1014.
20. Hessel, M.; Modayil, J.; Van Hasselt, H.; Schaul, T.; Ostrovski, G.; Dabney, W.; Horgan, D.; Piot, B.; Azar, M.; Silver, D. Rainbow: Combining Improvements in Deep Reinforcement Learning. Thirty-Second AAAI Conference on Artificial Intelligence, 2018.
21. Dulac-Arnold, G.; Mankowitz, D.; Hester, T. Challenges of Real-World Reinforcement Learning. *arXiv preprint arXiv:1904.12901* **2019**.
22. Haj-Ali, A.; Ahmed, N.K.; Willke, T.; Gonzalez, J.; Asanovic, K.; Stoica, I. Deep Reinforcement Learning in System Optimization. *arXiv preprint arXiv:1908.01275* **2019**.
23. Buesing, L.; Weber, T.; Racaniere, S.; Eslami, S.; Rezende, D.; Reichert, D.P.; Viola, F.; Besse, F.; Gregor, K.; Hassabis, D.; others. Learning and Querying Fast Generative Models for Reinforcement Learning. *arXiv preprint arXiv:1802.03006* **2018**.
24. Schmidhuber, J. Deep Learning in Neural Networks: An Overview. *Neural Networks* **2015**, *61*, 85–117.
25. Graves, A. Generating Sequences With Recurrent Neural Networks. *arXiv preprint arXiv:1308.0850* **2013**.
26. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural computation* **1997**, *9*, 1735–1780.
27. Bishop, C.M. Mixture Density Networks **1994**.
28. Bellemare, M.G.; Naddaf, Y.; Veness, J.; Bowling, M. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research* **2013**, *47*, 253–279.
29. Duan, Y.; Chen, X.; Houthooft, R.; Schulman, J.; Abbeel, P. Benchmarking Deep Reinforcement Learning for Continuous Control. International Conference on Machine Learning, 2016, pp. 1329–1338.
30. Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; Zaremba, W. OpenAI Gym. *arXiv preprint arXiv:1606.01540* **2016**.
31. Riiser, H.; Vigmostad, P.; Griwodz, C.; Halvorsen, P. Commute Path Bandwidth Traces from 3G Networks: Analysis and Applications. Proceedings of the 4th ACM Multimedia Systems Conference, 2013, pp. 114–118.
32. Hill, A.; Raffin, A.; Ernestus, M.; Gleave, A.; Kanervisto, A.; Traore, R.; Dhariwal, P.; Hesse, C.; Klimov, O.; Nichol, A.; Plappert, M.; Radford, A.; Schulman, J.; Sidor, S.; Wu, Y. Stable Baselines. https://github.com/hill-a/stable-baselines, 2018.
33. Akiba, T.; Sano, S.; Yanase, T.; Ohta, T.; Koyama, M. Optuna: A Next-generation Hyperparameter Optimization Framework. Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2019.
34. Bergstra, J.S.; Bardenet, R.; Bengio, Y.; Kégl, B. Algorithms for Hyper-Parameter Optimization. Advances in Neural Information Processing Systems, 2011, pp. 2546–2554.

35. Doshi-Velez, F.; Kim, B. Towards A Rigorous Science of Interpretable Machine Learning. *arXiv preprint arXiv:1702.08608* **2017**.