Significant Sampling for Shortest Path Routing: A Deep Reinforcement Learning Solution

Significant Sampling for Shortest Path Routing: A Deep Reinforcement Learning Solution

Yulin Shao[†], *Student Member, IEEE*, Arman Rezaee[†], *Student Member, IEEE*, Soung C. Liew, *Fellow IEEE*, Vincent W. Chan, *Life Fellow IEEE, Fellow OSA*

Abstract

Significant sampling is an adaptive monitoring technique proposed for highly dynamic networks with centralized network management and control systems. The essential spirit of significant sampling is to collect and disseminate network state information when it is of significant value to the optimal operation of the network, and in particular when it helps identify the shortest routes. Discovering the optimal sampling policy that specifies the optimal sampling frequency is referred to as the significant sampling problem. Modeling the problem as a Markov Decision process, this paper puts forth a deep reinforcement learning (DRL) approach to tackle the significant sampling problem. This approach is more flexible and general than prior approaches as it can accommodate a diverse set of network environments. Experimental results show that, 1) by following the objectives set in the prior work, our DRL approach can achieve performance comparable to their analytically derived policy ϕ' – unlike the prior approach, our approach is model-free and unaware of the underlying traffic model; 2) by appropriately modifying the objective functions, we obtain a new policy which addresses the never-sample problem of policy ϕ', consequently reducing the overall cost; 3) our DRL approach is robust to different stochastic variations of the network environment – it can provide good solutions under complex network environments where analytically tractable solutions are not feasible.

Index Terms

Network monitoring, shortest path routing, significant sampling, deep reinforcement learning.

I. Introduction

We are witnessing the dawn of a new technological era where new applications and devices have revolutionized every facet of our daily lives. Many of these applications produce and consume data at unprecedented rates with strict reliability and latency requirements [1], [2]. Not surprisingly, the bursty and unpredictable nature of their traffic can induce highly dynamic network environments that

[†]Authors have contributed equally to this work. Y. Shao and S. C. Liew are with the Dept. of Information Engineering, The Chinese University of Hong Kong ({sy016, soung}@ie.cuhk.edu.hk). A. Rezaee and V. W. S. Chan are with the Dept. of Electrical Engineering and Computer Science, MIT, ({armanr, chan}@mit.edu}).

threaten their own viability. Unencumbered operation of these applications requires rapid (re)actions by the Network Management and Control (NMC) system which itself depends on timely collection of network state information (NSI) [3], [4].

Our recent works focus on identifying the shortest path between a pair of communicating nodes in highly dynamic networks. In [5] we introduced a centralized routing protocol in which an NMC system monitors the network state (e.g., link/queuing delays) to decide which paths should be used to connect different origin-destination (OD) pairs. The monitoring process can be interpreted as a sampling process involving three steps: a) updating: the nodes in the network report their state (i.e. their most recent queuing delay over a period) to the NMC system; b) decision: the NMC system identifies the shortest path between each OD pair, and computes the next report time; c) dissemination: the NMC system disseminates the updated routing information and the next report time to all the nodes in the network.

A critical issue for this routing protocol is to determine the sampling frequency. Uniform sampling at high frequency can be burdensome and costly due to the excessive use of network transport and computational resources, while sparse sampling can lead to misconfiguration and suboptimal routing decisions [4]. Hence, judicious sampling of network states should be an essential feature of any pragmatic NMC system. In this context, we introduced the notion of significant sampling in [5], where the next report time is computed by the NMC from its cognitive understanding of the network states and short-term behavior of exogenous offered traffic. In a nutshell, the NMC will decide the next sampling time based on the likelihood that the optimal routes should be recomputed. The problem of discovering the optimal sampling policy for the cognitive NMC is referred to as the significant sampling problem.

In an effort to get an analytical solution to significant sampling, our work in [5] proposed the Ornstein-Uhlenbeck (OU) process as the underlying traffic/delay model and derived the optimal sampling policy when the parameters of the OU process are known to the NMC system. Despite its advantageous analytical nature, the work in [5] has a few shortcomings that we wish to address in this paper. First, the derived policy ϕ' suffers from the "never-sample" problem: when the cost of sampling is large, the policy ϕ' can instruct the NMC system to "never sample", which can be detrimental in the long run. Second, considering the bursty and unpredictable nature of traffic in dynamic networks, the assumed OU delay model is narrow in scope. A realistic and robust sampling policy should be able handle and smoothly transition through various operating regimes.

In this paper, we design an automated and model-free NMC system using deep reinforcement

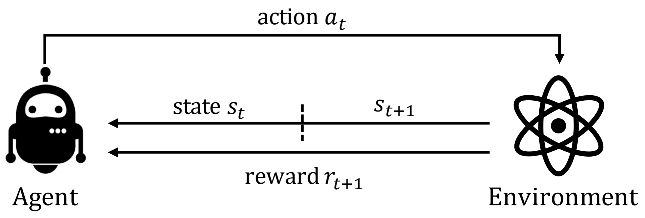


Figure 1: The interactions between agent and environment in reinforcement learning [6].

learning (DRL) techniques. As a refresher, we note that a salient feature of RL is "learning from interactions". As depicted in Fig. 1, the agent interacts with the environment in a sequence of time steps indexed by t. Given the observed state of the environment, s_t, the agent takes action a_t, which steers the environment to state s_{t+1} in the next time step. The environment then feedbacks a reward r_{t+1}, from which the agent can measure the quality of action a_t. A mapping between s_t and a_t is referred to as a policy function. The aim of the agent is generally to learn the optimal policy that maximizes the cumulative future rewards. The latest trend in RL research is to integrate the recent advances of deep learning [7] into the RL framework [8], [9]. RL that makes use of deep neural networks to approximate the optimal policy function – directly or indirectly – is referred to as deep reinforcement learning (DRL) [10]. DRL allows RL algorithms to be applied when the number of possible state-action pairs is enormous which inhibits traditional function approximators from accurately approximating the policy function.

The main contributions of this paper can be summarized as follows:

- We prove that the sampling policy ϕ' derived in [5] suffers from the never-sample problem when sampling cost is large. We characterize the never-sample region for the setup in [5].
- We put forth a DRL solution to the significant sampling problem. Our DRL solution has the advantage that it makes no assumption about the underlying traffic model, but learns to sample the network in such a way as to achieve optimal performance. We demonstrate the optimality of our DRL solution by using it to learn the policy ϕ'. Experimental results confirm that the learned policy matches the analytical results that were derived in [5].
- We design a multi-step look-ahead policy, based on our DRL solution, to address the never-sample problem of policy ϕ'. This new policy considers the long-term impact of a decision, and minimizes the cost rate over an infinite time horizon. Experimental results show that our new policy outperforms policy ϕ' by 39%, in terms of the average cost rate.
- Given the model-free nature of RL, we demonstrate that our DRL solution is robust to various network conditions. In this context, we extend the DRL framework beyond the work in [5] and provide solutions for general cases beyond the two-path OU process.

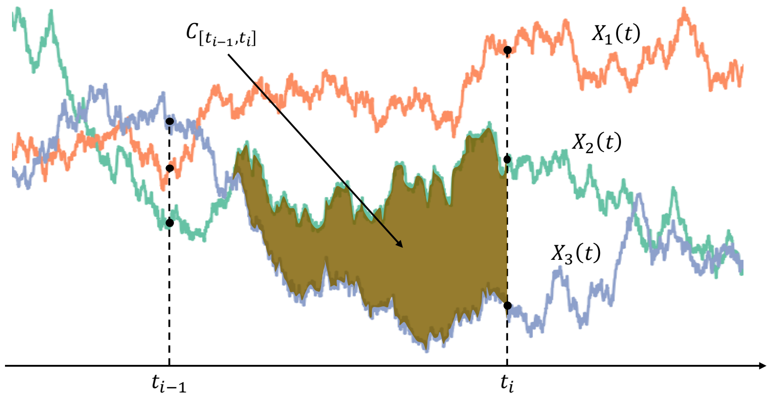


Figure 2: Cost of error, $C_{[t_{i-1},t_i]}$, between two consecutive sampling epochs t_{i-1} and t_i.

II. System Model

A. Significant Sampling

Let us consider an OD pair in a dynamic network, and assume that the OD pair is connected via N paths $\{P_n\!:\!n\!=\![N]\}$, where $[N] = 1, 2, \ldots, N$. The stochastically evolving weights of the N paths are denoted by $\{X_n(t)\!:\!n\!=\![N]\}$[1]. In shortest path routing, an NMC system will sample the weight of the N paths by orchestrating three distinct actions [5]: a) the nodes in each path report their weights to the NMC system; b) the NMC system identifies the path with the smallest weight and computes the next reporting/sampling time; c) the NMC disseminates the routing information and next report/sampling time to each nodes. As an example, suppose that the nodes on the N paths report[2] their weights to the NMC at a sampling epoch t_{i-1}. Given observation $\{X_n(t)\!:\!n\!=\![N], t \leq t_{i-1}\}$, the NMC will decide to route through $P_{n'}$, where $n' = \operatorname{argmin}_n X_n(t_{i-1})$. This routing information, along with the next sampling epoch, will be disseminated to all the nodes on each path. The same route will be used until the next sampling epoch t_i, at which point the NMC will use $\{X_n(t)\!:\!n\!=\![N], t \leq t_i\}$ to make new routing decisions. A fixed cost η is associated with one sampling operation, which captures the efforts required to collect and disseminate the NSI throughout the network[3].

[1]Usually, weight refers to delay (queuing delay in particular).

[2]At each sampling epoch, we assume that the nodes will report the trajectory of their evolving weights since the last sampling report to the NMC. For example, at t_i, the full realizations of $\{X_n(t) : n = [N], t_{i-1} \leq t \leq t_i\}$ will be provided to the NMC system, where t_{i-1} is the last sampling time.

[3]To be more specific, the cost of one sampling operation is composed of communication cost and cost of action. The communication cost is introduced in steps a) and c), and can be considered as constant. For example, the communication cost in the step a) can be fixed to one IP packet/node (considering a typical IP packet containing 1K Bytes, a node can send 500 real numbers to the NMC per update if we use 2 bytes to represent a real number). The cost of action is introduced by the computation in step b), computational cost of updating routing-tables, and other side-effects. We use η as a catch-all quantity to capture the overall cost of one sampling operation.

As illustrated in Fig. 2, if the shortest path changes between two sampling epochs t_{i-1} and t_i, we will incur an excess cost $C_{[t_{i-1},t_i]}$ given by

$$C_{[t_{i-1},t_i]} = \int_{t_{i-1}}^{t_i} X_{n'}(t) - \min_n X_n(t)\, dt,$$

where $n' = \operatorname{argmin}_n X_n(t_{i-1})$ is the shortest path observed at epoch t_{i-1}, and $\min_n X_n(t)$ denotes the weight of the "true" shortest path at time t.

Not surprisingly, continuous sampling of the stochastic process can reduce the cost of error to zero, but will result in an extremely high sampling cost. On the other hand, sparse sampling incurs a low sampling cost, at the expense of higher cost of error. The optimal tradeoff between sampling cost and cost of error can be described through a policy that specifies the best future sampling times based on the previous observations of the weight processes. Specifically, we write the policy at epoch t_{i-1} as a deterministic function that maps the history of weight processes $s(t_{i-1}) = \{X_n(t) : n = [N], t \leq t_{i-1}\}$ to the next sampling interval T_i:

$$T_i = \phi(s(t_{i-1}))$$

In particular, if the weight processes are Markovian (i.e., all the information is captured by the most recent observation, and prior history of the process is inconsequential for optimal decision making), we can design $s(t_{i-1})$ to be $s(t_{i-1}) = \{X_n(t_{i-1}) : n = [N]\}$.

The optimal policy ϕ^*, which specifies the optimal T_i^* at epoch t_{i-1}, minimizes the cost rate (i.e., cost per unit time) over an infinite time horizon, giving

$$\phi^* = \operatorname*{argmin}_{\phi} \lim_{L\to\infty} \frac{\sum_{l=i}^{L} \left(\eta + C_{[t_{l-1}, t_{l-1}+\phi(s(t_{l-1}))]}\right)}{\sum_{l=i}^{L} \phi(s(t_{l-1}))} \tag{1}$$

We refer to the problem of discovering the optimal policy ϕ^* as the "significant sampling" problem for shortest path routing.

B. Prior Solution

Our prior work [5] derived the analytical solutions for the significant sampling problem under three assumptions:

1) $N = 2$, i.e., the OD pair is connected via only two paths with weights $X_1(t)$ and $X_2(t)$. Since there are only two paths, identifying the shorter path is equivalent to identifying the sign of the difference process $X(t) = X_1(t) - X_2(t) : X_1(t) \lesseqgtr X_2(t)$ if and only if $X(t) \lesseqgtr 0$.

2) Weight processes $X_1(t)$ and $X_2(t)$ are independent Ornstein-Uhlenbeck (OU) processes with the same mean value. This means $X(t)$ is a zero-mean OU process.
   The OU process assumption follows the heavy-traffic results of Halfin and Whitt [11], which show that when service times are exponentially distributed, the sequence of appropriately normalized queue lengths will converge to the OU process. Mathematically, OU process is the only non-trivial continuous random process that is simultaneously Gaussian, Markovian, and stationary; properties that are often necessary in obtaining analytical solutions. An OU process OU(t) parameterized by $\{\mu_0, \theta_0, \sigma_0\}$ is governed by the following stochastic differential equation:

$$d\text{OU}(t) = \theta_0(\mu_0 - \text{OU}(t))\, dt + \sigma_0\, dW(t)$$

   where μ_0 is the long-term mean of the process, $\theta_0 > 0$ is the mean-reversion speed, and σ_0 is the volatility. This expression consists of two parts. The first part, $\theta_0(\mu_0 - \text{OU}(t))\, dt$, captures the mean reverting behavior, which is akin to a force that pulls the process towards its long-term mean. The second part, $\sigma_0\, dW(t)$, is a standard Wiener process scaled by volatility factor σ_0. This second part acts as additive noise and counteracts the mean reversion. Hence, the OU process can be roughly described as noisy oscillations around μ_0.
3) Directly derive policy ϕ^* is quite tricky, as both numerator and denominator in (1) can increase unboundedly. In [5], we approximate the optimal policy ϕ^* by

$$\phi' = \underset{\phi}{\operatorname{argmin}} \frac{\eta + \mathbb{E}\left[C_{[t_{i-1}, t_{i-1}+\phi(s(t_{i-1}))]}\right]}{\phi(s(t_{i-1}))} \tag{2}$$

   Compared with ϕ^*, policy ϕ' is myopic and minimizes the cost rate in $[t_{i-1}, t_i]$, and takes no account of the actions after t_i. We name the policy ϕ' a "one-step look-ahead policy".

Given the above assumptions, [5] focuses on sampling the difference process $X(t)$, an OU process parameterized by $\{\mu = 0, \theta, \sigma\}$[4]. Since OU process is Markovian, we can simplify $s(t_{i-1})$ to $s(t_{i-1}) = X(t_{i-1})$. The policy ϕ' can then be written as

$$\phi'(X(t_{i-1}) = x) = T_i'\Big|_{X(t_{i-1})=x} = \underset{T_i>0}{\operatorname{argmin}} \frac{\eta + \mathbb{E}\left[C_{[t_{i-1}, t_{i-1}+T_i]}|X(t_{i-1}) = x\right]}{T_i}, \tag{3}$$

[4]If both $X_1(t)$ and $X_2(t)$ are OU processes parameterized by $\{\mu_0, \theta_0, \sigma_0\}$, then $X(t)$ is an OU process parameterized by $\{\mu = 0, \theta = \theta_0, \sigma = \sqrt{2}\sigma_0\}$.

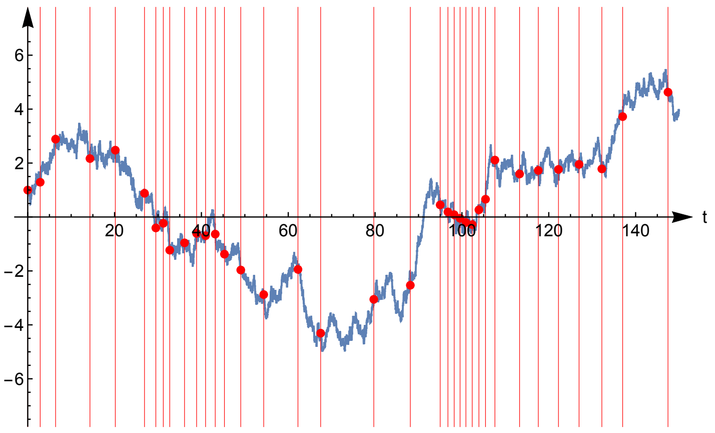


Figure 3: Applying the sampling policy ϕ' on a realization of an OU process with parameters $\{\mu = 0, \theta = 0.025, \sigma = 0.5\}$. The sampling cost $\eta = 0.1$.

where [5] further derived

$$\mathbb{E}\left[C_{[t_{i-1},t_{i-1}+T_i]}|X(t_{i-1})=x\right] = \frac{x\left(\mathrm{e}^{-\theta T_i}-1\right)}{\theta} + \frac{\sigma}{4\theta\sqrt{\theta\pi}}\int_0^{\mathrm{e}^{2\theta T_i}-1}\frac{\sqrt{y}}{(1+y)^{\frac{3}{2}}}\exp\left(-\frac{\theta x^2}{y\sigma^2}\right)dy$$
$$+\frac{x}{4\theta}\int_0^{\mathrm{e}^{2\theta T_i}-1}\frac{1}{(1+y)^{\frac{3}{2}}}\operatorname{erfc}\left(-\frac{\sqrt{\theta}x}{\sqrt{y}\sigma}\right)dy, \qquad (4)$$

The optimal sampling interval $T_i'|_{X(t_{i-1})=x}$ under this policy can then be computed numerically by solving the non-convex optimization in (3).

Fig. 3 depicts a realization of an OU process with parameters $\{\mu = 0, \theta = 0.025, \sigma = 0.5\}$. The result of setting the sampling cost to $\eta = 0.1$ and applying the sampling policy ϕ' is depicted in Fig. 3. As shown, the sampling frequency increases when $X(t)$ is close to zero (corresponding to times when $X_1(t) \approx X_2(t)$). On the other hand, when $X(t)$ is far from zero (i.e when $X_1(t)$ and $X_2(t)$ are far apart) the algorithm will adopt a sparse sampling strategy.

III. Limitations of Prior Solution

This section points out and elaborates two limitations of the analytical solution in [5].

Hard to generalize – The assumptions in [5] are meant to simplify the significant sampling problem so that analytical solutions can be derived. However, the simplified problem still leads to complicated expressions that are hard to interpret (e.g., Eq. (4)). Furthermore, extensions to more general cases (e.g., the N-path case, or non-OU processes) can be even more complex.

The never-sample problem – Policy ϕ' performs well when the sampling cost is small [5]. However, when the sampling cost is large, policy ϕ' instructs the NMC system to "never sample". The details of this never-sample problem can be understood through the following Proposition 1.

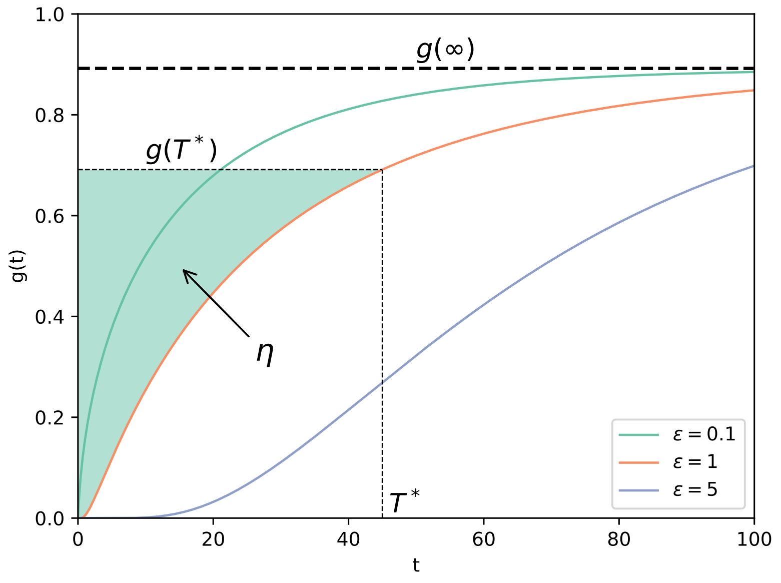


Figure 4: An intuitive explanation of policy ϕ'. The difference process $X(t)$ is a zero-mean OU process parameterized by $\{\mu = 0, \theta, \sigma\}$; $X(t') = \varepsilon$ is the sampled value at epoch t'; and $g(t)$ in Eq. (6) is the instantaneous cost rate for $t > t'$ conditioned on $X(t') = \varepsilon$.

Proposition 1. *Suppose $X(t)$ is a zero-mean OU process parameterized by $\{\mu = 0, \theta, \sigma\}$. At epoch t_{i-1}, we sample $X(t)$ and obtain $X(t_{i-1})$. Then, there exists a real number $\varepsilon \geq 0$ such that $\forall\ |X(t_{i-1})| \leq \varepsilon$, policy ϕ' will instruct the NMC to never sample the process if*

$$\eta > \int_0^\infty \sqrt{\frac{\sigma^2}{4\pi\theta}} - g(t)\, dt, \tag{5}$$

where

$$g(t) = \sqrt{\frac{\sigma^2(1-e^{-2\theta t})}{4\pi\theta}} e^{-\frac{\theta\varepsilon^2}{\sigma^2(e^{2\theta t}-1)}} - \frac{\varepsilon e^{-\theta t}}{2} \operatorname{erfc}\left(\sqrt{\frac{\theta\varepsilon^2}{\sigma^2(e^{2\theta t}-1)}}\right) \tag{6}$$

Proof. Assuming at some epoch t', we sample $X(t)$ and obtain $X(t') = \varepsilon$. The decision rule of policy ϕ' for the next sampling interval T^* can be described visually by Fig. 4 (see Appendix A for detailed derivations and analyses).

As shown in Fig. 4, the shaded area increases with T; the optimal sampling interval T^* indicated by policy ϕ' is the T that makes the shaded area equal to the sampling cost η. As a result, the policy ϕ' can instruct us to "never sample" if η is greater than the shaded area even when $T \to \infty$. This gives us the lower bound for η in (5). When (5) is satisfied, if the absolute value of any sample $X(t_{i-1})$ is less than or equal to ε, policy ϕ' will instruct the NMC to never sample, and the cost rate will converge to $g(\infty) = \sqrt{\frac{\sigma^2}{4\pi\theta}}$ as time passes.

□

It should be pointed out that "never sample" is not always suboptimal if our objective is to minimize the cost rate over an infinite time horizon, as in (1). A simple example is when $\eta \to \infty$, for which the optimal policy ϕ^* is to never sample. However, the following Proposition 2 further shows that policy ϕ' is guaranteed to be suboptimal if η is smaller than an upper bound.

Proposition 2. *Policy ϕ' is suboptimal in terms of minimizing the cost rate over an infinite time horizon if*

$$\int_0^\infty \sqrt{\frac{\sigma^2}{4\pi\theta}} - g(t)\, dt < \eta < \sqrt{\frac{\sigma^2}{4\pi\theta}} \frac{1}{\theta}. \tag{7}$$

In particular, sampling the weight process once at the equilibrium state can outperform ϕ'.

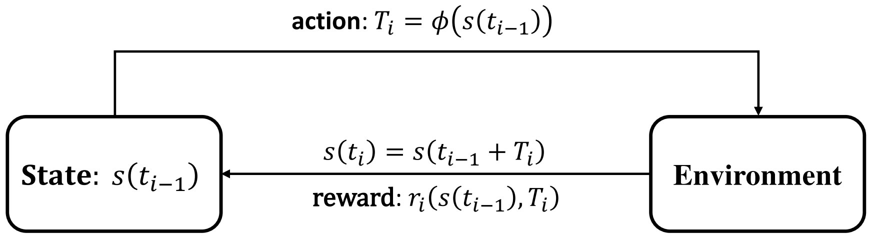
Proof. See Appendix A. □

To overcome the two limitations above, this paper considers another approach. In particular, we put forth a deep reinforcement learning (DRL) solution to solve the significant sampling problem. A DRL solution has the advantage that it makes no assumption about the weight processes (often referred to as the "model-free" property of DRL), and is thus more robust to various network conditions that give rise to different stochastic variations of $X_n(t)$. This enables us to remove the assumptions made in [5], and provide solutions to more general setting that go beyond the 2-path limitation and traffic modeling assumption (OU process). Moreover, we leverage this DRL framework to design a better policy that minimizes the cost rate over a long time horizon. This new policy, by planning farther beyond the immediate future, can effectively address the never-sample problem faced by policy ϕ'.

IV. A DRL Solution to Significant Sampling

A. Significant Sampling as an RL Problem

Let us start by transforming the original significant sampling problem to a reinforcement learning problem. The basic idea is that we can view the NMC system as an agent, and the underlying weight processes $\{X_n(t)\}$ as the environment. As shown in Fig. 5, the agent will maintain a ledger that contains the state of the environment, i.e., $s(t_{i-1}) = \{X_n(t) : n = [N], t \leq t_{i-1}\}$. Given state $s(t_{i-1})$, the agent's policy ϕ will output an action $T_i = \phi(s(t_{i-1}))$, which effectively specifies the next epoch for sampling the environment and thus the time for updating the ledger. At $t_i = t_{i-1} + T_i$, the agent will sample the environment, update the ledger with $s(t_i) = \{X_n(t) : n = [N], t \leq t_i\}$, and the cycle continues.



action: $T_i = \phi(s(t_{i-1}))$
State: $s(t_{i-1})$
$s(t_i) = s(t_{i-1} + T_i)$
reward: $r_i(s(t_{i-1}), T_i)$
Environment

Figure 5: Significant sampling as a reinforcement learning problem. An agent interacts with the environment to learn the optimal sampling policy.

The agent's ability to improve its actions depends on a feedback system that can assign values to various actions so that the agent can distinguish good actions from bad ones. This feedback is often expressed in terms of a *reward* function which indicates the reward associated with taking a particular action when the environment is in state $s(t_{i-1})$. For significant sampling, the environment will provide $\{X_n(t) : n = [N], t_{i-1} \leq t \leq t_i\}$ as feedback at the end of a time step, from which the agent computes a reward r_i (to be defined later) to evaluate the action T_i in this time step. We shall refer to a complete cycle from t_{i-1} to t_i as one "time step". The learning process will traverse many time steps, and each time step provides the agent with a new experience denoted by the quaternion $\{s(t_{i-1}), T_i, r_i, s(t_i)\}$.

1) Action-value function: In standard RL algorithms, the agent's objective at epoch t_{i-1} is to learn a policy that maximizes the future *return* R_i, a common definition of which is the sum of discounted future rewards

$$R_i = \sum_{l=i}^{\infty} \gamma^{l-i} r_l, \tag{8}$$

where $\gamma \in [0, 1]$ is a discounting factor. An action-value function (also known as the Q function) describes the maximum achievable return if the agent takes action T_i in state $s(t_{i-1})$, and then follows the optimal policy from then on. That is,

$$Q^*(s(t_{i-1}), T_i) = \max_{\phi} \mathbb{E}_{\text{env}} \left[R_i | s(t_{i-1}), T_i, \phi \right] = \mathbb{E}_{\text{env}} \left[r_i + \gamma \max_{\phi} Q^*(s(t_i), \phi(s(t_i))) \right], \tag{9}$$

where $Q^*(s(t_{i-1}), T_i)$ is often referred to as the Q value of action T_i in state $s(t_{i-1})$. This recursive form is known as the Bellman equation [6].

The action-value function is important in RL algorithms because it is a direct reflection of the optimal policy. That is, the action with the maximal Q value should be the action chosen by the optimal policy. On the other hand, if we know the action-value function, the optimal policy can be easily extracted by acting greedily (i.e., choose the action with the maximal Q value).

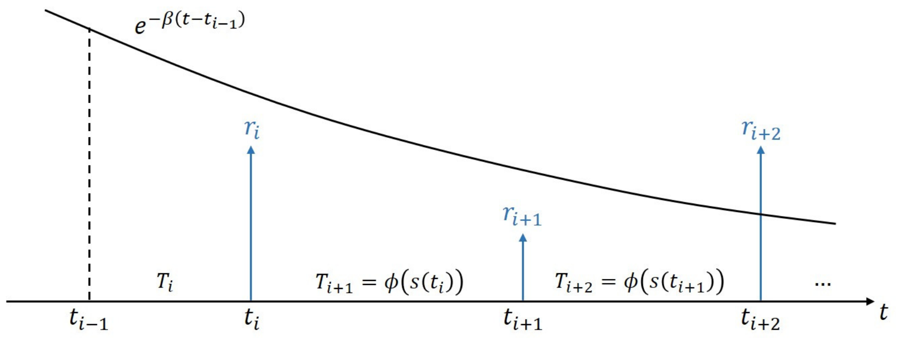


Figure 6: Illustration of exponentially discounting reward for significant sampling, in which the dynamics of the environment are continuous.

2) Learn policy ϕ': As an example, we can make use of the DRL framework to learn the policy ϕ'. To this end, the reward and return can be defined as

$$r_i = -\frac{\eta + C_{[t_{i-1},t_i]}}{T_i}, \qquad R_i = r_i. \tag{10}$$

The action-value function is then given by

$$Q^*(s(t_{i-1}), T_i) = \max_{\phi} \mathbb{E}_{\text{env}} \left[r_i \mid \phi \right], \tag{11}$$

As shown in Eq. (11), the policy ϕ' is myopic and only maximizes the one-step reward. Inspired by (8), we can design a farsighted significant sampling policy that considers future rewards, well beyond the immediate one-step future.

B. Action-value Function Design

In standard RL problems, the agent interacts with the environment in uniform time steps, and rewards are received at the end of time steps. The algorithm can then use the (discrete) γ-discounting to define the return as in (8). On the other hand, the durations of time-steps in significant sampling vary from time step to time step, and the dynamics of the environment are continuous. To address this issue, we propose a more appropriate exponentially discounting method for the computation of the return, instead of using the discrete γ-discounting.

As illustrated in Fig. 6, the agent takes action T_i in state $s(t_{i-1})$ and receives the feedback $\{X_n(t) : n = [N], t_{i-1} \le t \le t_i\}$ from the environment at t_i. Let us define

$$\varphi_i(t) = X_{n'_{i-1}}(t) - \min_n X_n(t), \qquad t_{i-1} \le t \le t_i,$$

where n'_{i-1} is the index of the shortest path observed at epoch t_{i-1}.

The exponentially discounted reward within $[t_{i-1}, t_i]$ can then be defined as

$$r_i = -\eta e^{-\beta T_i} - \int_{t_{i-1}}^{t_i} e^{-\beta(t-t_{i-1})} \varphi_i(t)\, dt \tag{12}$$

where $\beta > 0$ is a exponential discount factor. Note that $\varphi_i(t)dt$ is the cost incurred subsequent to t_{i-1} and is therefore discounted according to the time elapsed since t_{i-1}. Likewise, the next sampling cost η is incurred at time $t_i = t_{i-1} + T_i$ and is therefore discounted corresponding to T_i, the time elapsed since t_{i-1}.

Similarly, at t_i, the agent takes action $T_{i+1} = \phi(s(t_i))$ and receives the feedback $\{X_n(t) : n = [N], t_i \leq t \leq t_{i+1}\}$ from the environment at t_{i+1}. The exponentially discounted reward within $[t_i, t_{i+1}]$ is then given by

$$r_{i+1} = -\eta e^{-\beta T_{i+1}} - \int_{t_i}^{t_{i+1}} e^{-\beta(t-t_i)} \varphi_{i+1}(t)\, dt$$

assuming discounting begins at t_i. Future rewards within each time step can be computed in a similar fashion. Overall, the return due to the action at t_{i-1} can be defined as

$$R_i = r_i + e^{-\beta T_i} r_{i+1} + e^{-\beta(T_i + T_{i+1})} r_{i+2} + \cdots = r_i + \sum_{l=i}^{\infty} e^{-\beta \sum_{l'=i}^{l} T_{l'}} r_{l+1}. \tag{13}$$

Note that in the above, the discounting of the constituent rewards in R_i begins at time t_{i-1}, hence the additional discounting factors for r_{i+1}, r_{i+2}, $\cdots$. Essentially, return R_i is the "present value" of all future rewards, where a reward received t time units later is worth only $e^{-\beta t}$ times what it would be worth if it were received immediately. The discounting is introduced to prevent the return from going to infinity, as in (8).

The action-value function can then be rewritten in the following recursive form:

$$Q^*(s(t_{i-1}), T_i) = \max_{\phi} \mathbb{E}_{\text{env}} \left[R_i | s(t_{i-1}), T_i, \phi \right] = \mathbb{E}_{\text{env}} \left[r_i + e^{-\beta T_i} \max_{\phi} Q^*(s(t_i), \phi(s(t_i))) \right], \tag{14}$$

Given the Bellman-style equation in (14), we can leverage DRL algorithms to learn the action-value function, and extract the corresponding optimal policy in a greedy manner. We shall refer to this new policy, which maximizes the return in (13), as the "multi-step look-ahead policy".

Compared with the optimal policy ϕ^* in (1), the multi-step look-ahead policy maximizes the sum of the exponentially discounted rewards the agent receives over an infinite time horizon. This is a better approximation to ϕ^* than the one-step look-ahead policy ϕ', especially when the discount exponent β is made to be small.

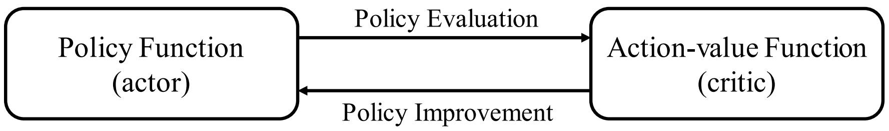


Figure 7: RL algorithms as a generalized policy iteration.

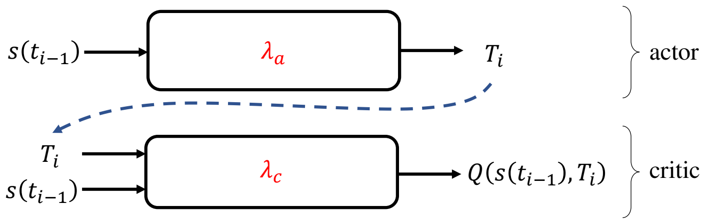


Figure 8: The neural networks in actor-critic RL. The actor approximates the policy function, and the critic approximates the optimal action-value function.

C. An Actor-critic Solution

In RL, the policy function is often referred to as the "actor", as it controls the behavior of the agent; the action-value function is often referred to as the "critic", as it measures the quality of an action. An actor-critic solution for RL problems can be described as a generalized policy iteration (GPI) [6] between the actor and the critic, as illustrated in Fig. 7.

The policy function (actor) and action-value function (critic) will be initialized randomly, and their evolution will progress through a series of policy evaluations and policy improvements. Policy evaluation makes the action-value function more consistent with the current policy in that it reflects better the returns produced by the current policy (using Bellman equation). Policy improvement, on the other hand, makes the policy greedier with respect to the current action-value function (using policy gradient [12]). These two processes alternate and progress until the actor converges to the optimal policy, and the critic converges to the optimal Q function.

Deep Deterministic Policy Gradient (DDPG) [13] is an actor-critic approach for MDP with continuous action space. This paper adapts DDPG to solve the RL problem associated with significant sampling. Specifically, we use two deep neural networks (DNNs), an actor network and a critic network, to approximate the policy function and the action-value function, respectively. Fig. 8 depicts the logical interaction of the actor and critic networks [14]. The actor, parameterized by λ_a, outputs an action T_i when the network state $s(t_{i-1})$ is used as its input, i.e., $T_i = \Psi_a(s(t_{i-1}); \lambda_a)$. The critic network, parameterized by λ_c, outputs a Q-value estimation $Q(s(t_{i-1}), T_i)$ when the state-action pair, $(s(t_{i-1}), T_i)$, is used as its input, i.e., $Q(s(t_{i-1}), T_i) = \Psi_c(s(t_{i-1}), T_i; \lambda_c)$.

These two DNNs are initialized randomly. As learning continues, we use the accumulated experiences to train both DNNs, as follows [14].

Suppose we have an experience $\{s(t_{i-1}), T_i, r_i, s(t_i)\}$.

- The critic is updated using the Bellman-style equation in (14). Specifically, given the state-action pair $\{s(t_{i-1}), T_i\}$, we update parameters λ_c in the direction that minimizes the mean square error (MSE) loss between the target Q value $r_i + e^{-\beta T_i}\Psi_c(s(t_i), \Psi_a(s(t_i);\lambda_a);\lambda_c)$ and the Q value $\Psi_c(s(t_{i-1}), T_i;\lambda_c)$ estimated by the current critic network:

$$-\nabla_{\lambda_c}\mathcal{L}_c = -\nabla_{\lambda_c}\left[r_i + e^{-\beta T_i}\Psi_c(s(t_i), \Psi_a(s(t_i);\lambda_a);\lambda_c) - \Psi_c(s(t_{i-1}), T_i;\lambda_c)\right]^2. \tag{15}$$

  Updating λ_c in the negative gradient direction given by (15) makes the output of the critic network closely match the target Q value. This allows the critic network to approximate the Q function evermore closely, resulting in more accurate policy evaluations.

  In practice, directly using (15) to update the critic usually leads to divergence of the Q function approximation. This is because the target Q value and the estimated Q value in (15) are highly correlated (as they are computed using the same critic network – note: the second term in the target Q is computed using the current critic network as well). To tackle this instability, people often use separate actor and critic networks (as opposed to using the current actor and critic networks in the second component of the target Q above) to compute the target Q value (see (17) below). This is known as “fixed Q-target” [8].

- The actor is trained using policy gradient. That is, for state $s(t_{i-1})$, we update parameters λ_a in the direction that maximizes the action-value function (i.e., the output of the critic):

$$\nabla_{\lambda_a}Q^{\lambda_c} = \nabla_{\lambda_a}\Psi_c(s(t_{i-1}), \Psi_a(s(t_{i-1});\lambda_a);\lambda_c). \tag{16}$$

  Updating λ_a in the gradient direction given by (16) makes the actor greedier with respect to the current critic. This is a process of policy improvement.

Following the practice of DQL [8] and DDPG [14], we also employ the ideas of experience replay in the implementation. The goal of experience replay is to remove the temporal correlations between successive experiences, such that the experiences used for training are less dependent. To this end, instead of learning experiences online, we put the accumulated experiences in a FIFO buffer of size M (experiences will be placed into and removed from this FIFO buffer continuously), and randomly sample experiences from the FIFO to train the DNNs.

D. Algorithm

The pseudocode for the actor-critic solution to significant sampling is given in Algo. 1.

Algo. 1: Significant sampling with Deep Reinforcement Learning.

```
Initialization:
    Initialize a FIFO of size $M$ for experience replay.
    Initialize an actor network and a critic network randomly with parameters $\lambda_a$ and $\lambda_c$.
    Initialize a target actor network and a target critic network with parameters $\lambda_{a'}$ and $\lambda_{c'}$.
    Let $\lambda_{a'} = \lambda_a$ and $\lambda_{c'} = \lambda_c$.
    Set mini-batch size to $K$, evaluation cycle to $B$, time step $i = 1$, training step $j = 1$.
    Set resampling window to $t_w$, the number of artificial experiences to $V$, and parameter $G$.
    In epoch $t_0$, all the nodes report their weights, obtain $s(t_{i-1})$.
while 1 do
    New experience:
    Feed $s(t_{i-1})$ into the actor network for action $T_i = \Psi_a\left(s(t_{i-1}); \lambda_a\right)$.
    In epoch $t_i = t_{i-1} + T_i$:
        All the nodes report their weights in $t \in [t_{i-1}, t_i]$.
        Compute reward $r_i$ by Eq. (12), and store experience $\{s(t_{i-1}), T_i, r_i, s(t_i)\}$ into FIFO.
        $i = i + 1$
    Artificial experiences:
    $v = 1$.
    while $v \leq V$ do
        Select a random starting epoch $t'_{v-1}$ in $[t_i - t_w, t_i]$,
        Feed state $s(t'_{v-1})$ into the actor network for $T'_v = \Psi_a\left(s(t'_{v-1}); \lambda_a\right)$.
        if $t'_{v-1} + T'_v < t_i$ then
            Compute reward $r'_i$ by Eq. (12).
            Store experience $\{s(t'_{i-1}), T'_i, r'_i, s(t'_i)\}$ into FIFO.
            $v = v + 1$.
        end
    end
    DNN training:
    if $i * (V + 1) > M/5$ then
        Randomly sample $G$ mini-batches from FIFO, a mini-batch has $K$ experiences.
        for each mini-batch do
            Update parameters $\lambda^c$ (critic) in the direction given by Eq. (17).
            Update parameters $\lambda^a$ (actor) in the direction given by Eq. (18).
            Update the target networks by Eq (19).
        end
        if $mod(j, B) == 0$ then
            Evaluate the performance of current actor (policy) network on the evaluation trajectory.
        end
        $j = j + G$.
    end
end
```

1) Initialization: As shown, we start by initializing a FIFO buffer of size M (for experience replay); the actor and critic networks with parameters λ_a and λ_c; and the target actor and critic networks with parameters $\lambda_{a'}$ and $\lambda_{c'}$ (for calculating target Q value). In particular, the parameters of target networks are initialized to $\lambda_{a'} = \lambda_a$ and $\lambda_{c'} = \lambda_c$. The initial sampling time is set to t_0, where $s(t_0)$ is given.

2) Experience collection: The next phase is "experience collection". At the beginning of a time step (i.e., at t_{i-1}), the agent feeds the current observation $s(t_{i-1})$ into the actor network, which outputs action T_i. Note that this action is bad early on, but will improve as learning progresses (to guarantee exploration, Gaussian noise is added to the raw output of the actor [14]). At the end of the time step (i.e., at t_i), the agent computes a reward r_i from the environment's feedback (see (12)). Overall, one time step provides the agent with one experience $\{s(t_{i-1}), T_i, r_i, s(t_i)\}$. All experiences will be stored in the FIFO for experience replay.

A challenge in learning the optimal policy for significant sampling is the inefficiency with which experiences are collected. Note that, one time step lasts for T_i time units, yet it gives the agent only one experience, hence is quite inefficient. This raises the following question: besides the collected experience, is there a way for the agent to generate more experiences artificially? We answer this question affirmatively by noting that in each time step, the continuously evolving weights $\{X_n(t) : n = [N]\}$ in interval $[t_{i-1}, t_i]$ are collected and reported to the NMC. As a result, at epoch t_i, all past weights $\{X_n(t) : n = [N], t \leq t_i\}$ are in fact known to the agent.

In this context, besides this new experience, the agent will resample the past and generate $\underline{\text{artificial experiences}}$. Specifically, each time a new experience is collected, the agent will randomly select V starting epochs in period $[t_i - t_w, t_i]$, where t_w is a resampling window[5]. For each starting epoch $\{t'_{v-1} : v = [V]\}$, the agent will feed $s(t'_{v-1})$ into the current actor network for action T'_v. It can then compute reward r'_v which constitutes an artificial experience $\{s(t'_{v-1}), T'_v, r'_v, s(t'_v)\}$. This enables the agent to collects $V + 1$ experiences during each time step, V of which are artificial.

3) DNN training: When the number of experiences is larger than $M/5$ (M is the buffer size), the agent starts to train the DNNs by mini-batch gradient descent [6]. We will randomly sample G mini-batches from the FIFO, with each mini-batch containing K experiences.

For each mini-batch, we have $\{s(t_{k-1}), T_k, r_k, s(t_k)\}, k \in \{i_1, i_2, \ldots, i_K\}$,

- We update parameters λ_c in the direction that minimizes the mean square error (MSE) loss

$$-\nabla_{\lambda_c} \frac{1}{K} \sum_{k \in \{i_1, \ldots, i_K\}} \left[r_k + e^{-\beta T_k} \Psi_{c'}(s(t_k), \Psi_{a'}(s(t_k); \lambda_{a'}); \lambda_{c'}) - \Psi_c(s(t_{k-1}), T_k; \lambda_c) \right]^2, \quad (17)$$

  Unlike (15), the target Q value in (17) is computed using a target actor network and a target critic network.

[5]In this paper, t_w is set to be a large value (e.g., 2000 time units, much larger than the duration of one time step) because we consider stationary environment. For non-stationary environments, the choice of t_w depends on the speed at which the environment varies, because new environment requires the agent to "forget" about the past and "relearn" from the latest observations. More discussions can be found in our conference paper [15].

- Given state $s(t_{i-1})$, we update parameters λ_a in the direction that maximizes the output of the critic (i.e., the Q value) by sampled policy gradient, giving

$$\begin{aligned}\nabla_{\lambda_a} Q^{\lambda_c} &\approx \frac{1}{K} \sum_{k\in\{i_1,\dots,i_K\}} \nabla_{\lambda_a} \Psi_c\left(s(t_{k-1}), T; \lambda_c\right)\Big|_{T=\Psi_a(s(t_{k-1});\lambda_a)} \\ &= \frac{1}{K} \sum_{k\in\{i_1,\dots,i_K\}} \nabla_T \Psi_c\left(s(t_{k-1}), T; \lambda_c\right) \nabla_{\lambda_a} \Psi_a\left(s(t_{k-1}); \lambda_a\right) \end{aligned} \tag{18}$$

- Each time the actor and critic networks are updated, the parameters of target networks will be updated using soft replacement [14]:

$$\lambda_{a'} = \rho\lambda_a + (1-\rho)\lambda_{a'}, \qquad \lambda_{c'} = \rho\lambda_c + (1-\rho)\lambda_{c'} \tag{19}$$

4) Performance monitoring: Finally, to monitor and illustrate the continuous improvements of the learned policy, we evaluate the performance of the policy/actor network every B training steps. The evaluation is performed on an $\underline{\text{independent}}$ trajectory of $X_n(t)$ (this evaluation set is completely independent from the sample path of the process that were observed during the training process).

V. Experimental Results

This section presents experimental results of the actor-critic DRL solution to significant sampling. Our main goals in this study are threefold: 1) to confirm that our DRL approach can learn the one-step policy ϕ' by adopting a corresponding one-step action-value function; 2) to show that, by adopting a multi-step look-ahead action-value function, our DRL approach can learn the multi-step policy that addresses the "never-sample" problem associated with the policy ϕ', and this policy is more optimal than the policy ϕ' in terms of minimizing long-term cost rate; 3) to show that our DRL approach can provide effective solutions for more general environment with more than two paths, and with weight-evolution processes beyond the OU process.

A. Learn the One-step Look-ahead Policy

This experiment makes use of the DRL approach to learn the one-step look-ahead policy ϕ'. The importance of the analytical solutions in (3) is that they allow us to benchmark the performance of our DRL solution against the theoretical result. A comparable performance will provide us with the foundation to extend our objective and setup beyond what was presented in [5] with a reasonable degree of confidence in its ability to perform close to optimal.

Table I: Hyper-parameter settings for the one-step look-ahead policy.

Description	Symbol	Value
FIFO size	M	32768
Mini-batch size	K	128
Resampling window	t_w	2000
# artificial experiences	V	31
# mini-batches per training	G	5
Maximum sampling interval	$T_{\max}$	40

Let us begin by assuming the same system setup as in [5]. Specifically, the OD pair is connected via two paths, and the evolving weights of both paths are represented by independent OU processes with the same mean value. Thus, the difference process $X(t)$ is a zero-mean OU process, and we can focus on sampling $X(t)$. The analytically derived policy ϕ' in (3) will serve as the benchmark.

To learn the one-step look-ahead policy, we can set the action-value function to be (11). That is, the objective of the agent is simply to maximize the one-step reward rate. The MSE loss in (17) can then be simplified to

$$-\nabla_{\lambda_c}\mathcal{L}_c = -\nabla_{\lambda_c}\frac{1}{K}\sum_{k\in\{i_1,\ldots,i_K\}}\left[\frac{r_k}{T_k} - \Psi_c(s(t_{k-1}),T_k;\lambda_c)\right]^2, \tag{20}$$

Compared with (17), the target value in (20) is simply r_k/T_k, and hence the target networks can be removed in this experiment.

Since the OU process is Markovian, we can set $s(t_{i-1}) = X(t_{i-1})$, i.e., the state of the environment is a real number. The actor and critic networks are designed to be fully-connected (feedforward) neural networks. The actor network consists of five fully connected layers, and the numbers of neurons in the five layers are $\{1, 32, 64, 32, 1\}$. The three hidden layers use rectifier non-linearity as activation functions, and the output layer uses hyperbolic tangent (tanh) non-linearity as the activation function. The output of the actor network u takes value in $(-1, 1)$ and is mapped to the sampling interval $T_i = T_{\max} * (u + 1)/2$, where $T_{\max}$ is the maximum sampling interval. The architecture of the critic network is the same as the actor network except that a) the input layer has two neurons instead of one; b) the output layer uses no activation function, and directly produces the Q value given a state-action pair (See Fig. 8 for additional reference).

The hyper-parameter settings are listed in Table I. We used Adam optimizer for training the neural network training, and set the learning rates for the actor and critic networks to 0.0005 and 0.001, respectively.

We generate two independent trajectories of $X(t)$ for training and evaluation purposes respectively. As in [5], the sampling cost is set to $\eta = 0.1$ and the OU parameters for $X(t)$ are chosen to be $\{\mu = 0, \theta = 0.025, \sigma = 0.5\}$. The duration of the training trajectory is 2×10^5 time units, and the duration of the evaluation trajectory is 2×10^4 time units.

Over the course of training, for every $B = 100$ training steps, we assess the performance of the actor network on the evaluation trajectory. Specifically, let us denote by ϕ_j the sampling policy learned by the actor network after j training steps. We will apply the policy ϕ_j on the whole evaluation trajectory, and compute the cost rate by

$$\overline{c}_j = \frac{\sum_{L_j}^{l=1} (\eta + C_{[t_{l-1}, t_l]})}{\sum_{L_j}^{l=1} T_l}, \tag{21}$$

where L_j is the number of samples of policy ϕ_j when applied on the whole evaluation trajectory, and the sampling epochs on the evaluation trajectory are $\{t_i : i = [L_j]\}$.

Fig. 9(a) presents the cost rate as a function of training steps in the RL process. As a benchmark, we also plot the minimum achievable cost rate (0.0492) which is obtained by applying the analytically derived one-step look-ahead policy given by (3) on the evaluation trajectory. We can see that the cost rate achieved by the learned policy decreases as training continues and eventually approaches the optimal cost. The learned policies at different times of the learning process are plotted in Fig. 9(b-e). The x-axis is the observed value of the weight process $X(t)$, and the y-axis is the corresponding action T_i given by the learned policy at that training step. The benchmark, again, is the analytically derived one-step look-ahead policy. As shown, the actor network will learn the optimal policy after enough training steps.

B. Learn the Multi-step Look-ahead Policy

Subsection V-A confirms that the actor-critic DRL approach can learn the one-step look-ahead policy derived in [5]. This is an important achievement in and of itself because the DRL system is model-free. Specifically, unlike the analysis giving rise to the analytical solution, the agent in DRL is unaware that the underlying delay model is an OU process and yet “learns” to obtain a near-optimal sampling policy.

This subsection evaluates the multi-step look-ahead policy based on the actor-critic framework. We have repeated the experiment in subsection V-A assuming a small sampling cost followed by assuming a large sampling cost.

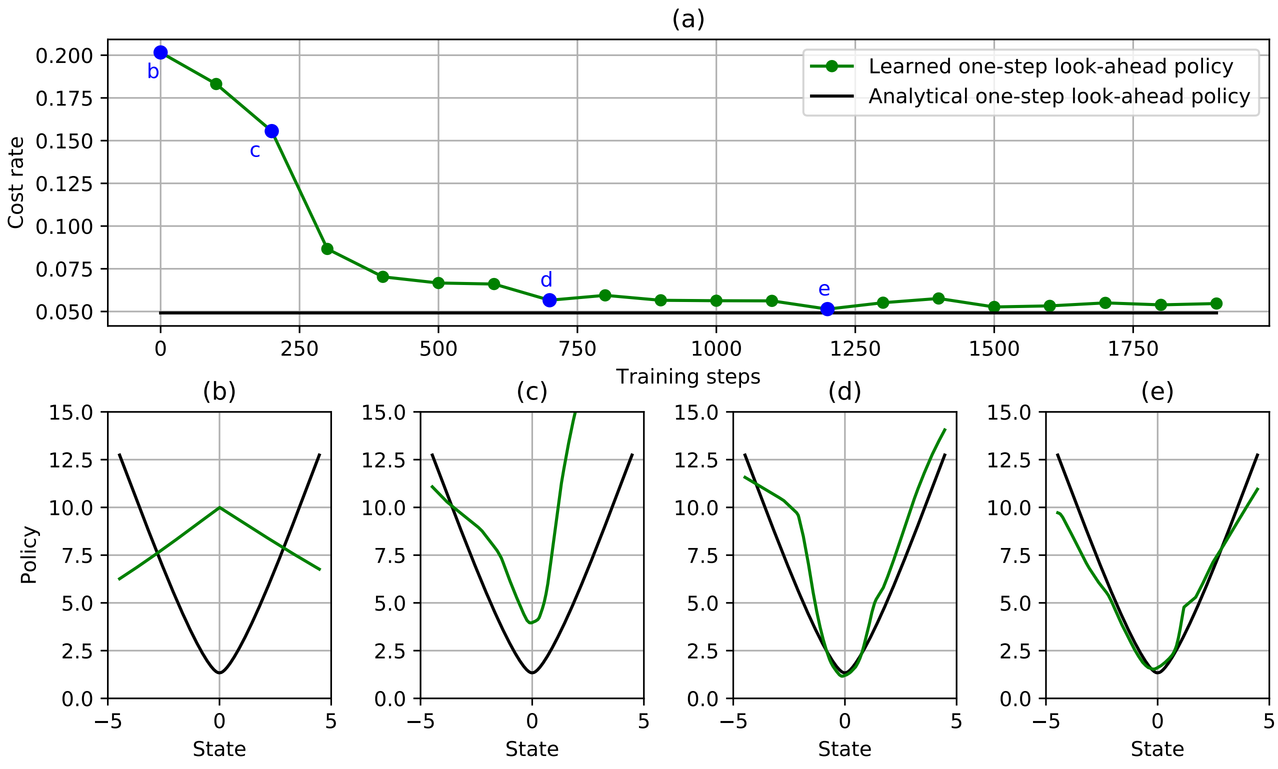


Figure 9: The cost rate achieved on the evaluation trajectory, and the policy improvements in the RL process: (a) cost rate achieved on the evaluation trajectory versus the number of training steps; (b) the learned policy of actor at point b; (c) the learned policy of actor at point c; (d) the learned policy of actor at point c; (e) the learned policy of actor at point e.

Fig. 10 shows the performance of the one-step and multi-step look-ahead policy when $\eta = 0.1$. The performance curves for one-step look-ahead policy, both analytical and learned, are reproduced from Fig. 9 (the difference is that the y-axis is in log scale). As shown, with $\eta = 0.1$, the multi-step look-ahead policy achieves only minor gain over the one-step look-ahead policy. This indicates that the one-step look-ahead policy is a good policy when η is small in terms of minimizing the long-term cost rate. This is not the case, however, for more general η.

In the second experiment, we use a larger sampling cost $\eta = 15$. Notice that $\eta = 15$ falls into the region $12.2 < \eta < 35.68$ given by (7) if we set $\varepsilon = 0.1$. Thus, if the absolute value of one sample is less than or equal to 0.1, the one-step look-ahead policy will instruct us to never sample, and the cost rate will eventually converge to $g(\infty) = \sqrt{\frac{\sigma^2}{4\pi\theta}}$.

Fig. 11 shows the cost rates achieved by the learned one-step and multi-step look-ahead policies. The analytical cost rate for one-step look-ahead policy is $g(\infty) \approx 0.89$ (i.e., the dashed straight line in Fig. 11). We note that the learned one-step look-ahead policy is better than the analytical results. This is because an “artificial” maximum sampling interval T_{max} is set in the implementation

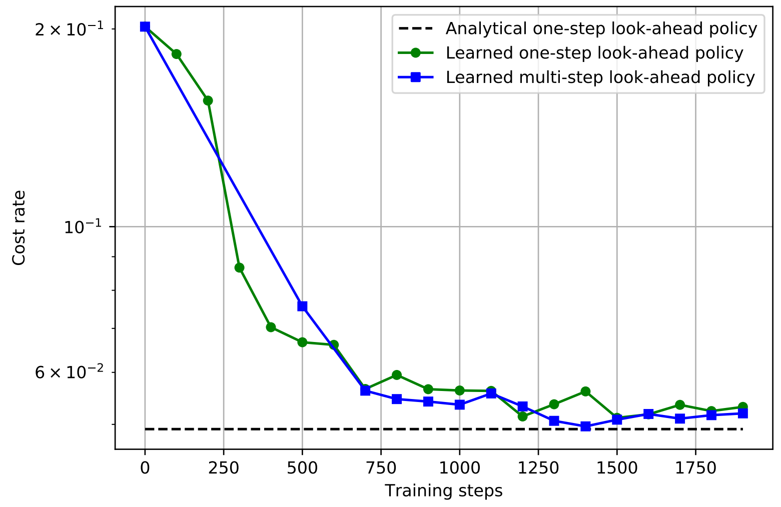


Figure 10: The cost rate achieved on the evaluation trajectory when the sampling cost $\eta = 0.1$.

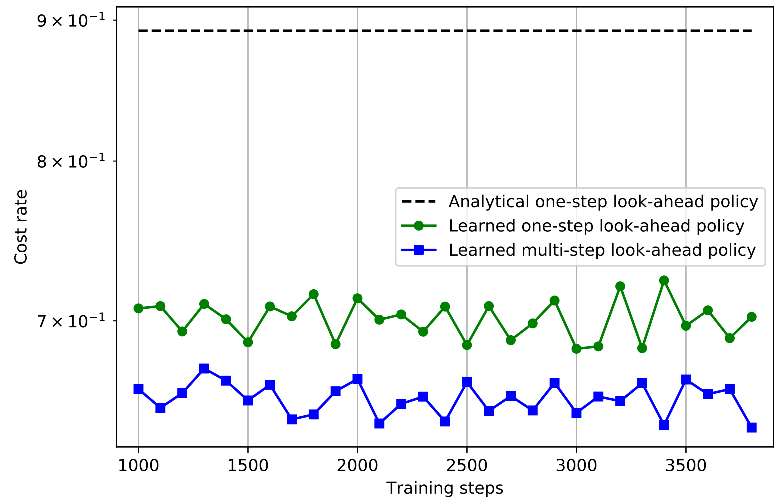


Figure 11: The cost rate achieved on the evaluation trajectory when the sampling cost $\eta = 15$.

(recall that the output of the actor DNN is $[0, T_{\max}]$, where $T_{\max} = 120$ in this experiment). Thus, if the absolute value of one sample is less or equal to 0.1, the action chosen by the one-step look-ahead policy is 120 rather than infinity. This is confirmed in Fig. 12, where the learned policies are plotted. As shown, for the one-step look-ahead policy, the learned policy is consistent with our prediction. When a sampled value is close to 0, the action given by the learned one-step look-ahead policy reaches the maximal sampling interval. On the other hand, for the multi-step look-ahead policy, the learned policy matches with our intuition of a good policy, i.e., when $X(t)$ is close to zero, sampling should be more frequent; when $X(t)$ is far away from zero, sampling should be more sparse. The never-sample problem is well addressed by our multi-step look-ahead policy.

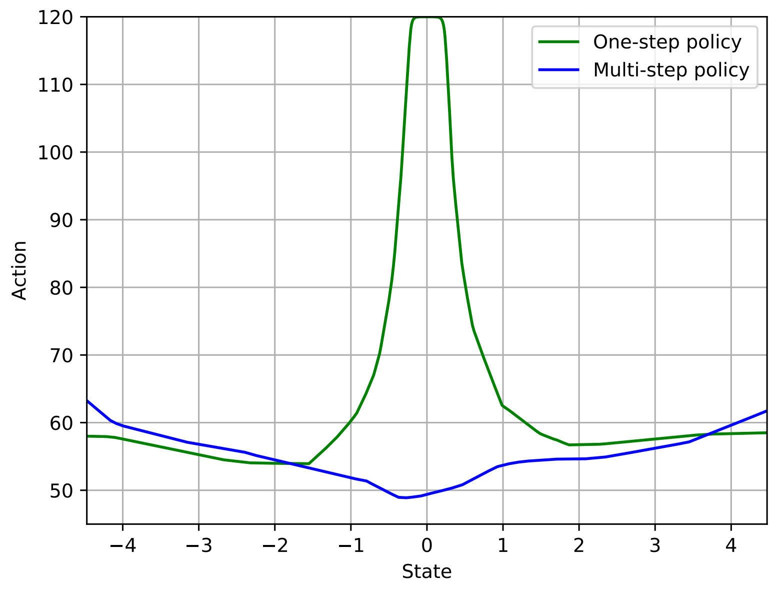


Figure 12: The one-step and multi-step look-ahead policies. The sampling cost $\eta = 15$.

Overall, the learned multi-step policy achieves the best long-term cost rate. Considering the average cost rate achieved on the evaluation trajectory, our multi-step policy outperforms the analytical one-step policy, and the learned one-step policy by 39% and 8%.

C. Extensions to More General Environments

Our previous discussions focused on the same system setup as in [5], i.e., the OD pair is connected via only two paths, and the weight processes of both paths are OU processes. Thanks to the model-free property of RL, an advantage of our DRL approach is that we can provide solutions for more general system setups with only slight modifications to the algorithm. This subsection will study two generalizations to the system setup in [5].

1) N paths and OU process: This experiment removes the $N = 2$ assumption. We assume there are N paths and the evolving weight of each path is an OU process. In particular, we choose $N = 3$ in the experiment.

The state of the environment is defined to be $s(t_{i-1}) = \{X_n(t_{i-1}) : n = [N]\}$. That is, the state input of the actor and critic networks is a vector of length N. We can then increase the number of neurons in the input layer of the actor and critic networks to N and $N + 1$, respectively. The hyper-parameters are the same as that in Table I. The three OU processes are generated using the same parameter set $\{\mu = 0, \theta = 0.025, \sigma = 0.3\}$, and the sampling cost $\eta = 0.1$. We use the DRL framework to learn the multi-step look-ahead policy in the implementation.

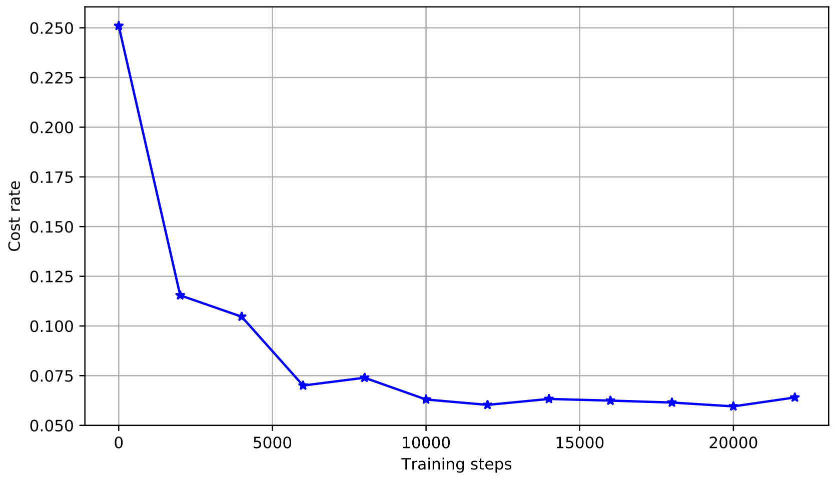


Figure 13: The cost rate achieved on the evaluation trajectory in the RL process. There are three paths, and we use the DRL framework to learn the multi-step look-ahead policy.

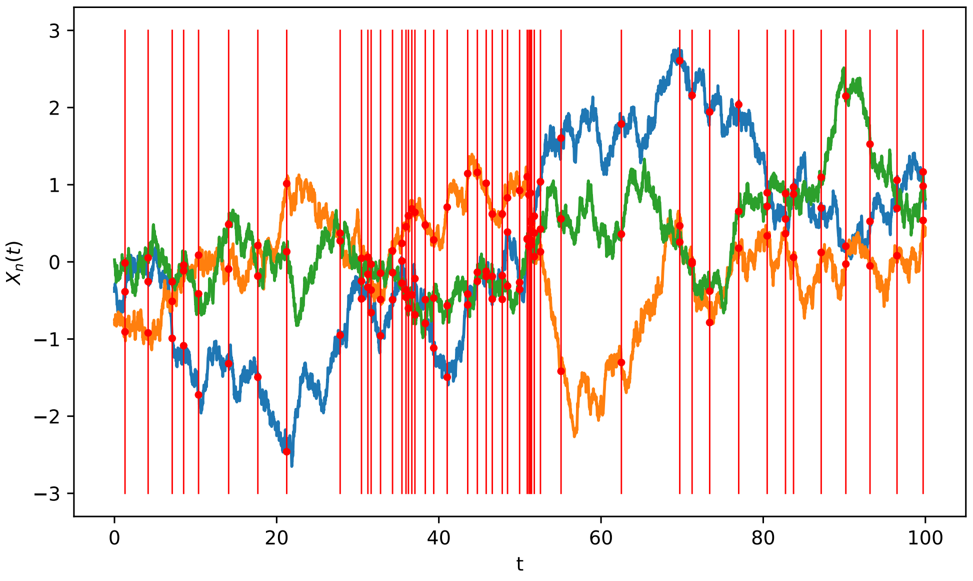


Figure 14: The result of applying the learned policy (after 25000 training steps) on a realization of $\{X_n(t) : n = [3]\}$.

Fig. 13 shows the cost rate achieved by the multi-step look-ahead policy on the evaluation trajectory. To evaluate the learning results, we apply the learned policy (after 25000 training steps) on a realization of $X_n(t)$, and plot the sampling results in Fig. 14. The learned policy matches our intuition of a good policy: it samples more frequently when the values of $\{X_n(t) : n = [3]\}$ are close, and more sparsely when one of the three weights is much smaller than the other two paths.

2) Two paths and weight processes with memory: In previous sections, we showed that the DRL framework can solve the significant sampling problem when the underlying weight process is taken to be an OU process. A favorable property of the OU process is that it is Markovian. Evolution of such an environment is completely captured through the latest sample, and hence we can define the state of the environment as $s(t_{i-1}) = \{X_n(t_{i-1}) : n = [N]\}$.

Unlike the OU process, real-world weight processes may exhibit complex interdependencies over time and have memory. This suggests that optimal sampling of such processes depends not only on the latest sample, but also on the history of process. To address this issue, one may define the state of the environment as $s(t_{i-1}) = \{X_n(t) : n = [N], t \leq t_{i-1}\}$, that is, we feed the full history of the weight process into the DNNs for decision making. However, such an approach will require an incredibly large and potentially infinite state space due to the continuous nature of $X_n(t)$, and is thus impractical. Hence, we seek to reduce the state space required for the optimal sampling of general processes. This section combines the following two schemes to achive this goal:

a) We characterize each weight process, $X_n(t)$, in the frequency domain, and use the ledger to represent $X_n(t)$ as a real vector obtained by uniformly sampling the process (as per Nyquist sampling theorem [16]). This allows us to reduce/compress the information contained in the continuous process/signal.
b) We assume the decision depends only on a finite period of recent history, and truncate early uniformly sampled points before this period.

Denote by $\boldsymbol{X}(t_{i-1}) = \{X_n(t_{i-1}) : n = [N]\}$ the N samples at a sampling epoch t_{i-1}. The state of the environment can then be defined as (22) following the above two schemes.

$$s(t_{i-1}) = \left\{ \boldsymbol{X}\left(t_{i-1} - \frac{L-1}{f_{\text{max}}}\right), \boldsymbol{X}\left(t_{i-1} - \frac{L-2}{f_{\text{max}}}\right), \ldots, \boldsymbol{X}\left(t_{i-1} - \frac{1}{f_{\text{max}}}\right), \boldsymbol{X}(t_{i-1}) \right\}, \qquad (22)$$

where f_{max} is the uniform sampling rate, and the history before $t_{i-1} - (L-1)/f_{\text{max}}$ is considered irrelevant to decision making. In a nutshell, state $s(t_{i-1})$ is composed of the samples at L most recent sampling epochs, if we sample the history of $X_n(t)$ uniformly at rate f_{max}.

The uniform sampling rate f_{max} is designed to be $f_{\text{max}} = \max\{f_1, f_2, \ldots, f_N\}$, where f_n is the uniform sampling rate for the n-th path. To determine f_n, we first transform $X_n(t)$ to the frequency domain. If $X_n(t)$ is band-limited, f_n can be set to the Nyquist rate of $X_n(t)$ (twice its bandwidth). If $X_n(t)$ is non-band limited as most real-world signals/weight processes, we will pass $X_n(t)$ through a low-pass filter and truncate high frequency components in the stopband of the filter (empirically, the spectrum of most practical process centers around DC, and progressively gets smaller and smaller as frequency increases). Then, the filtered $X_n(t)$ is a band-limited signal, and we set f_n to be its Nyquist rate.

To confirm the proposed schemes are effective, we conduct the following experiment, in which an artificial weight process with memory is used as input. In this example, we assume the OD pair is connected via two paths, and we focus on sampling the difference process $X(t)$. In particular, we artificially generate $X(t)$ by adding a periodicity (i.e., a rectangular wave) to an OU process,

$$X(t) = \text{OU}(t) + \sqrt{\frac{2\sigma^2}{\theta}} \sum_i \text{Rect}(t - iD_R), \tag{23}$$

where $\text{OU}(t)$ is an OU process parameterized by $\{\mu = 0, \theta = 0.025, \sigma = 0.5\}$; the rectangular wave $\text{Rect}(t) = \text{sgn}(t), -D_R/2 \leq t < D_R/2$ (D_R is the duration of one period, we set $D_R = 20$ in the experiment). The rectangular wave is scaled by the two standard deviations of the OU process. A realization of $X(t)$ is shown in Fig. 16. Compared with OU process, this new process undergoes sudden changes, and hence is one of the most challenging processes for the DRL framework to learn a good sampling policy[6].

To learn a good policy for the weight process in (23), the first step is to determine an uniform sampling rate as in (22). Following scheme a), we set the uniform sampling rate $f_{\text{max}} = 1$ sample per unit time (sampling at this rate, the dominant spectrum of $X(t)$ has already been captured). As a result, at an epoch t_{i-1}, we will uniformly sample the history of $X(t)$, and set the state of the environment at t_{i-1} to be a collection of L most recent uniform samples, i.e., $s(t_{i-1}) = \{X(t_{i-1} - l), l = L-1, L-2, \ldots, 0\}$. We will monitor the performance of the learned policy for different values of L.

In the experiment, the sampling cost is set to $\eta = 0.1$, and we let the DRL framework learn the multi-step look-ahead policy. All experimental setups are the same as Section IV except for the DNN architecture. The shape of the actor is now $\{L, 64, 128, 64, 32, 1\}$, where the L input neurons are used to represent state $s(t_{i-1})$. The shape of the critic is $\{L+1, 64, 128, 64, 32, 1\}$.

First, we set $L = 1$. This means the actor network need to determine the optimal sampling interval T_i from state $s(t_{i-1}) = X(t_{i-1})$. We then assess the learned policy on an evaluation trajectory every $B = 4000$ training steps, and plot the achieved cost rate in Fig. 15. Then, we increase L. The final performance of the learned policy improves as L increases until $L = 10$. No

[6]We successfully experimented with various processes with memory, including ARMA, ARIMA, filtered OU, etc. but the process in Eq. (23) was the best candidate to showcase the ability of our algorithm, as the sudden changes of the rectangular wave present a challenging environment.

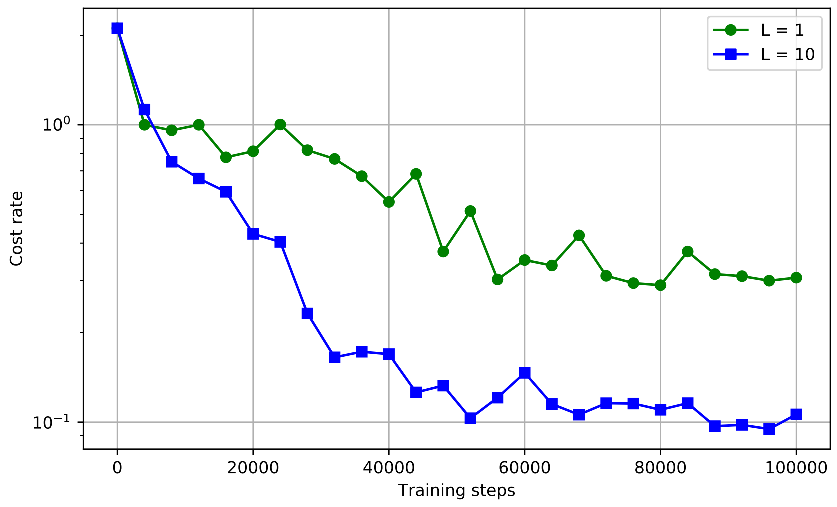
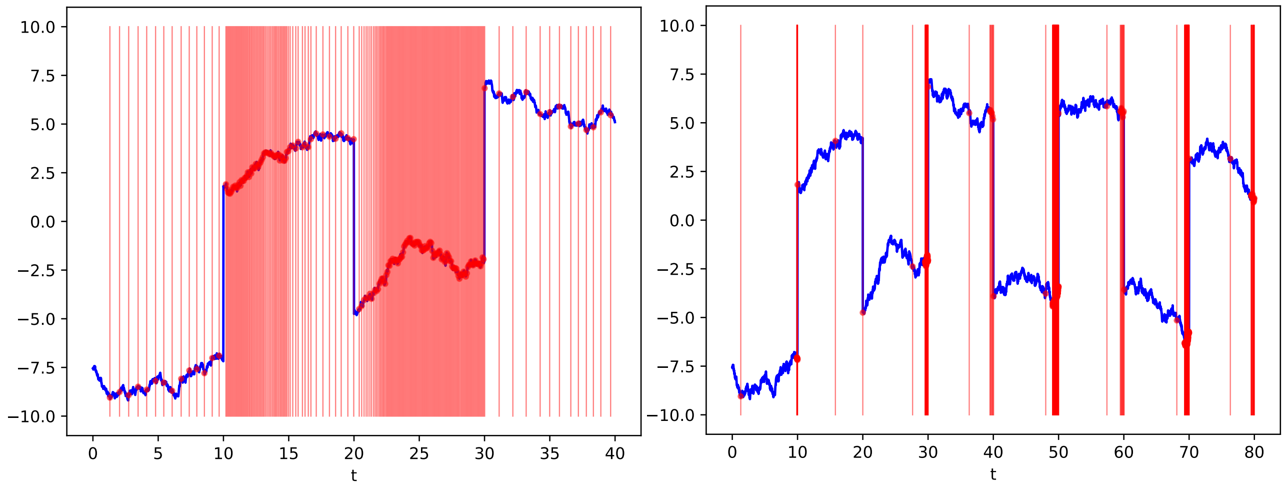


Figure 15: The cost rate achieved on the evaluation trajectory given different input length L.



(a) $L = 1$

(b) $L = 10$

Figure 16: Evaluation of the learned policy when $L = 1$ and $L = 10$.

further improvements were observed for $L > 10$. Note that when $L = 10$, the history captured in state $s(t_{i-1})$ is half the period of the rectangular wave. Overall, choosing $L = 10$ helps improve the average cost rate (at convergence) by a factor of 3.2 in comparison to when $L = 1$.

To evaluate the learning results, we apply the learned policy (after 80000 training steps) on a realization of $X(t)$, and plot the sampling results in Fig. 16. As can be seen, the learned policy is bad when $L = 1$. This means the actor network is unable to make a good decision given only the latest observation of $X(t)$. On the other hand, when $L = 10$, the leanred policy samples the weight process mostly at the half-cycle of the rectangular wave where the sign of the rectangular wave changes.

VI. Conclusion

An important feature of any high-performance, resilient, and scalable networking platform is its ability to monitor the state of the network and reconfigure network resources and protocols accordingly. This is even more important in highly dynamic networks considering the bursty and unpredictable nature of the traffic.

This paper studied the significant sampling problem in highly dynamic networks with a centralized Network Management and Control (NMC) system. A sampling operation of the NMC system is a process of collecting network state information (NSI), identifying the shortest path(s) for different origin-destination (OD) pairs, and disseminating routing information to the networks. A sampling policy specifies the next sampling time based on the cognitive understanding of the network states at this moment. The problem of significant sampling is to discover the optimal sampling policy that effectively balances the cost of sampling and the cost of error associated with mis-identifying the shortest path until the next sampling time.

This paper put forth a deep reinforcement learning (DRL) solution to the significant sampling problem. Modeling the problem as a Markov Decision Process (MDP), we treated the NMC system as an agent that samples/measures the state of various network elements in order to identify the shortest path(s) and make optimal decisions on the sampling frequency. The agent periodically receives a reward commensurate with the quality of its actions. The decision on when to sample progressively improve as the agent learns the relationship between the sampling frequency and the reward function.

Benchmarked against the policy ϕ' derive in prior work [5], our DRL solution showed its ability to learn the target policy without a need for an explicit knowledge of the traffic model or its parameters. Moreover, on the basis of the DRL solution, this paper designed a new multi-step look-ahead policy to address the never-sample problem faced by policy ϕ'. Experimental results showed that average cost rate of policy ϕ' is 1.39 times of our new policy. Another advantage of our DRL solution is that it is robust to various network conditions and stochastic variations of traffic, thanks to the model-free property of RL. Two extensions to the system setup in [5] were discussed in the paper. In one extension, we considered the case of an N-path OU process. In the other extension, we considered a much more complex traffic model that is a mixture of OU process and periodic rectangular wave. It was confirmed that our DRL approach can provide good solutions to both extensions.

Appendix A

A. The One-step Look-ahead Policy

Suppose $X(t)$ is a zero-mean OU process parameterized by $\{\mu = 0, \theta, \sigma\}$. Without loss of generality, we sample $X(t)$ at time $t = 0$, and obtain $X(0) = \varepsilon$. Then, following the one-step look-ahead policy ϕ' in (3), the optimal sampling interval T^* can be obtained from

$$T^* = \arg\min_{T>0} \frac{\eta + \mathbb{E}[C_{[0,T]}|X(0)=\varepsilon]}{T} \triangleq \arg\min_{T>0} h(T),$$

where $C_{[0,T]}$ is the accumulated cost of error in $[0, T]$. Set $\frac{d\,h(T)}{dT} = 0$, we have

$$\begin{aligned} \frac{d\,h(T)}{dT} &= \frac{d\,\mathbb{E}[C_{[0,T]}|X(0) = \varepsilon]}{dT} T - (\eta + \mathbb{E}[C_{[0,T]}|X(0) = \varepsilon]) = 0 \\ \eta &= \frac{d\,\mathbb{E}[C_{[0,T]}|X(0) = \varepsilon]}{dT} T - \mathbb{E}[C_{[0,T]}|X(0) = \varepsilon]. \end{aligned} \tag{24}$$

To compute $\mathbb{E}[C_{[0,T]}|X(0)=\varepsilon]$, we can write $C_{[0,T]}$ as $C_{[0,T]} = \int_0^T \varphi(t)dt$, in which $\varphi(t)$ is the instantaneous cost of error at epoch $t \in [0, T]$, and is given by $\varphi(t) = -X(t)\mathbb{1}_{(-\infty,0)}(X(t))$ (that is, when $X(t) \leq 0$, $\varphi(t) = -X(t)$; else, $\varphi(t) = 0$).

Thus,

$$\mathbb{E}[C_{[0,T]}|X(0) = \varepsilon] = \int_0^T \mathbb{E}[\varphi(t)|X(0) = \varepsilon]\,dt \triangleq \int_0^T g(t)\,dt,$$

and (24) can be refined as

$$\eta = \frac{d \int_0^T g(t)\,dt}{dT} T - \int_0^T g(t)\,dt = Tg(T) - \int_0^T g(t)\,dt. \tag{25}$$

We then compute $g(t)$. Given $X(0) = \varepsilon$, we have

$$X(t) = \varepsilon e^{-\theta t} + \frac{\sigma}{\sqrt{2\theta}} e^{-\theta t} W(e^{2\theta t} - 1), \quad t > 0, \tag{26}$$

where $W(e^{2\theta t} - 1)$ is a time-scaled Winner process, $W(e^{2\theta t} - 1) \sim \mathcal{N}(0,\, e^{2\theta t} - 1)$. This means $X(t) \sim \mathcal{N}(\mu_t,\, \sigma_t^2)$ at time $t > 0$, where $\mu_t = \varepsilon e^{-\theta t}$ and $\sigma_t^2 = \frac{\sigma^2}{2\theta}(1 - e^{-2\theta t})$.

Function $g(t)$ can then be computed as

$$\begin{aligned} g(t) &= \int_{-\infty}^{0} -x f_{X(t)}(X(t) = x)dx = \frac{\sigma_t}{\sqrt{2\pi}} e^{-\frac{\mu_t^2}{2\sigma_t^2}} - \frac{\mu_t}{2} \operatorname{erfc}\left(\frac{\mu_t}{\sqrt{2\sigma_t^2}}\right) \\ &= \sqrt{\frac{\sigma^2(1 - e^{-2\theta t})}{4\pi\theta}} e^{-\frac{\theta\varepsilon^2}{\sigma^2(e^{2\theta t} - 1)}} - \frac{\varepsilon e^{-\theta t}}{2} \operatorname{erfc}\left(\sqrt{\frac{\theta\varepsilon^2}{\sigma^2(e^{2\theta t} - 1)}}\right) \end{aligned} \tag{27}$$

In fact, $g(t)$ is the expected instantaneous cost of error at t given $X(0) = \varepsilon$. It describes the transient behavior of $X(t)$ after sampling. Two key observations from (27) are

- $g(t)$ is an increasing function of t, and a decreasing function of ε.
- $g(0) = 0$ and $g(\infty) = \sqrt{\frac{\sigma^2}{4\pi\theta}}$ are constants irrelevant to ε.

B. Intuitive Explanations of the One-step Look-ahead Policy

Given (25) and (27), the one-step look-ahead policy ϕ' can be illustrated by Fig. 4, where we plot $g(t)$ when $\varepsilon = 0.1$, 1, and 5, respectively.

Let us focus on $\varepsilon = 1$ case, the optimal sampling interval T^* given by the one-step look-ahead policy is the T that makes the shaded area equal to the sampling cost η. This implies, the one-step look-ahead policy can instruct us to "never sample" if

$$\eta > \int_0^\infty g(\infty) - g(t)\, dt. \tag{28}$$

In this case, the shaded area will always be less than η as $t \to \infty$, and the system will never sample again. Moreover, since $g(t)$ is a decreasing function of ε, the policy basically tells us to never sample again if the absolute value of one sample is less or equal to ε. As time goes, the cost rate of the one-step look-ahead policy converges to $g(\infty) = \sqrt{\frac{\sigma^2}{4\pi\theta}}$.

Eq. (28) gives us the lower bound for η. The next subsection gives an upper bound for η to guarantee that "never sample" is suboptimal. We will show that, if we sample once at the equilibrium state long time later than $t = 0$, and then never sample again, the achieved cost rate can beat the one-step look-ahead policy.

C. Sample at the Equilibrium State

When comes to the equilibrium state, an OU process $X(t) = \frac{\sigma}{\sqrt{2\theta}} e^{-\theta t} W(e^{2\theta t})$,

Assuming we sample at $t = \tau$ in the equilibrium state, and obtain $X(\tau) = x_\tau$, then $x_\tau \sim \mathcal{N}(0, \sigma_\tau^2)$, where $\sigma_\tau^2 = \frac{\sigma^2}{2\theta}$. Given observation $X(\tau) = x_\tau$, the value of $X(t)$ at epoch $\tau + t$ $(t > 0)$ follows (26) and is a Gaussian random variable. We have

- $X(\tau + t) \sim \mathcal{N}(\mu_{\tau+t}, \sigma_{\tau+t}^2), \quad t > 0$, where $\mu_{\tau+t} = x_\tau e^{-\theta t}$ and $\sigma_{\tau+t}^2 = \frac{\sigma^2}{2\theta}(1 - e^{-2\theta t})$.
- The instantaneous cost of error $\varphi(\tau + t) = -X(\tau + t)\mathbb{1}_{(-\infty,0)}(X(\tau + t))$.

Then, for $x_\tau \geq 0$, we have

$$\mathbb{E}[\varphi(\tau+t)|X(\tau)=x_\tau] = \int_{-\infty}^0 -x f_{X(\tau+t)}(X(\tau+t)=x)dx = \frac{\sigma_{\tau+t}}{\sqrt{2\pi}} e^{-\frac{\mu_{\tau+t}^2}{2\sigma_{\tau+t}^2}} - \frac{\mu_{\tau+t}}{2}\,\mathrm{erfc}\left(\frac{\mu_{\tau+t}}{\sqrt{2}\sigma_{\tau+t}}\right)$$

Averaged over all possible x_τ, $x_\tau \sim \mathcal{N}(0,\, \sigma_\tau^2)$,

$$\mathbb{E}[\varphi(\tau+t)] = \int_{-\infty}^{\infty} \mathbb{E}[\varphi(\tau+t)|X(\tau){=}x_\tau] f_{X(\tau)}\left(X(\tau)=x_\tau\right) dx_\tau = \sqrt{\frac{\sigma^2}{4\pi\theta}}(1-e^{-\theta t})$$

Thus, if we sample at $t=\tau$, and the never sample again, the overall cost from $t=\tau$ to $t=\infty$ will be $\eta + \int_0^\infty \mathbb{E}[\varphi(\tau+t)]dt$. Let $\eta + \int_0^\infty \mathbb{E}[\varphi(\tau+t)]dt < \int_0^\infty g(\infty)dt$, we have

$$\eta < \int_0^\infty [g(\infty) - \mathbb{E}[\varphi(\tau+t)]]dt = \sqrt{\frac{\sigma^2}{4\pi\theta}} \int_0^\infty e^{-\theta t} dt = \sqrt{\frac{\sigma^2}{4\pi\theta}} \frac{1}{\theta} \tag{29}$$

Combing (28) and (29) gives us Proposition 2.

References

[1] I. Lee and K. Lee, "The internet of things (IoT): applications, investments, and challenges for enterprises," *Business Horizons*, vol. 58, no. 4, pp. 431–440, 2015.

[2] M. Bennis, M. Debbah, and H. V. Poor, "Ultra reliable and low-latency wireless communication: tail, risk, and scale," *Proc. IEEE*, vol. 106, no. 10, pp. 1834–1853, 2018.

[3] V. W. Chan, "Optical flow switching networks," *Proc. IEEE*, vol. 100, no. 5, pp. 1079–1091, 2012.

[4] V. W. Chan, "Cognitive optical networks," in *IEEE Int. Conf. Commun. (ICC)*, IEEE, 2018.

[5] A. Rezaee and V. W. Chan, "Cognitive network management and control with significantly reduced state sensing," *IEEE Trans. Cognitive Commun. and Networking*, 2019.

[6] R. Sutton, A. Barto, and F. Bach, *Reinforcement learning: an introduction.* MIT Press, 2018.

[7] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[9] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[10] Y. Li, "Deep reinforcement learning: an overview," *arXiv:1701.07274*, 2017.

[11] S. Halfin and W. Whitt, "Heavy-traffic limits for queues with many exponential servers," *Operations research*, vol. 29, no. 3, pp. 567–588, 1981.

[12] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, pp. 1057–1063, 2000.

[13] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Int. Conf. Mach. Learning (ICML)*, 2014.

[14] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv:1509.02971*, 2015.

[15] Y. Shao, A. Rezaee, S. C. Liew, and V. W. Chan, "Significant sampling for shortest path routing: a deep reinforcement learning solution," in *IEEE Global Commun. Conf. (Globecom)*, IEEE, 2019.

[16] C. E. Shannon, "Communication in the presence of noise," *Proc. Inst. Radio Eng*, vol. 371, 1934.