FINDING OPTIMAL POLICY FOR QUEUEING MODELS: NEW PARAMETERIZATION

Trang H. Tran[1], **Lam M. Nguyen**[2], **Katya Scheinberg**[1]
[1] School of Operations Research and Information Engineering, Cornell University, Ithaca, NY, USA
[2] IBM Research, Thomas J. Watson Research Center, Yorktown Heights, NY, USA
htt27@cornell.edu, LamNguyen.MLTD@ibm.com, katyas@cornell.edu

ABSTRACT

Queueing systems appear in many important real-life applications including communication networks, transportation and manufacturing systems. Reinforcement learning (RL) framework is a suitable model for the queueing control problem where the underlying dynamics are usually unknown and the agent receives little information from the environment to navigate. In this work, we investigate the optimization aspects of the queueing model as a RL environment and provide insight to learn the optimal policy efficiently. We propose a new parameterization of the policy by using the intrinsic properties of queueing network systems. Experiments show good performance of our methods with various load conditions from light to heavy traffic.

1 Introduction

The optimal control problem of queueing networks has many important applications in real life, including communications networks, transportation, and manufacturing systems [Prabhu, 1986, Perkins and Kumar, 2000, Srikant and Ying, 2013, Nguyen and Stolyar, 2017, Dai and Shi, 2019]. The main goal is to efficiently determine the optimal control policy, especially when the network is heavily loaded. In certain cases, such a policy can be derived analytically as a function of system parameters, but even in those simple cases, since system parameters are rarely known in practice, applying a learning approach is often preferable [Liu et al., 2019]. Queueing networks can be modeled effectively as a Reinforcement Learning (RL) environment, where an agent chooses actions in reaction to the environment and receives feedback to learn a good policy. RL has been a very active area recently, with the success of well-known algorithms such as AlphaGo [Silver et al., 2016] and OpenAIFive [OpenAI, 2018] and is expected to have tremendous impact on real-life applications, including autonomous driving, dynamic treatment regimes in healthcare and robotics manipulation. Despite its practical success, there is not enough understanding of the algorithmic frameworks used in this rich environment, especially, the optimization process.

The main goal of this paper is to investigate the key steps of the optimization process in model-free RL using a particular practical and yet tractable application of simple queuing network control. There have been recent works applying Reinforcement Learning framework to the queueing systems control [Liu et al., 2019, Dai and Gluzman, 2020]. However, these papers use a model-based approach where the agent estimates the system parameters in order to create a model of the system. In this paper, we are interested in investigating performance of the model-free framework on this problem, without explicitly modeling and utilizing the system dynamics.

We choose to consider a simple parallel queueing system with one server, exponential interarrival and service times. The optimal policy for this system is a well known threshold-type and can be computed based on the service time - holding cost ratio of each queue. A RL method is expected to learn this policy without the explicit knowledge of the service and arrival rates and possibly even without knowledge of the holding costs by simply maximizing a reward function (or minimizing the cost function), which is computed by simulating the queueing system over a set of parameters that define the policy. There are many choices which influence the efficiency of the optimization process and the effects of most of these choices are poorly explored in the literature. For example, while the classical REINFORCE estimator [Williams, 1992] has been used widely in many RL applications, it is not clear how it behaves in comparisons with other gradient estimators, and in particular, on queueing models. In addition, the choice of reward

functions and policy parameterizations have a significant effect on the nature of the resulting optimization problem and thus on the optimization process. Here we investigate this effect by considering some alternative choices for model-free RL applied to queueing systems.

Contributions:

- We investigate a simple linear policy parametrization which can be used within a model-free RL framework and show that it is able to approximate the optimal threshold with arbitrary accuracy. Based on the representation and approximation of the (optimal) priority policy, we then propose a new logarithm-scale parameterization which changes the problem scale and allows for smoother simpler optimization landscapes.
- We compared two different methods of constructing gradient approximation: the REINFORCE estimator Williams [1992] and the Finite Difference gradient estimator [Schlögl, 2018, Tu et al., 2019]. We implement the two gradient estimators with an adaptive line-search algorithm which is robust to gradient estimators and whose performance can be easily derived as a function of the cost and the error of the gradient estimators.
- Finally, we show that logarithm-scale parameterization behaves similarly or better than linear-scale parameterization regardless of the choice of the gradient estimator.

Related Works.

Reinforcement Learning. Reinforcement learning techniques have gained significant popularity recently for a large variety of applications, while the general RL problem setting remains difficult without specificity of the environment and the structure of the objective [Agarwal et al., 2019, Sutton and Barto, 2018]. The framework is generally described as a Markov decision process (MDP), with a particular transition probability function and a reward mechanism that returns the feedback after the agent takes an action. The typical algorithm for solving an MDP is via policy iteration or value iteration [Pashenkova et al., 1996], however, this approach is often not suitable in learning applications when the dimension of the problem is large. Another approach is to estimate the state-value function or the value function such as Q-learning [Watkins and Dayan, 1992] and its variants (e.g. Deep Q-learning (DQN) [Mnih et al., 2013, 2015], and double Q-learning [Hasselt et al., 2016]). However, it has been observed that learning the state-value function is not efficient when the action space is large or even infinite.

Parameterized Policy and Optimization. More popular recent approach to RL is to learn the policy directly as a parameterized function which maps the state space to a distribution over the action space. Neural networks are typical candidates for such parameterized functions because they have shown efficiency in many machine learning applications [Sutton and Barto, 2018, Agarwal et al., 2019, Dai and Gluzman, 2020]. However, in this paper we show that a simpler linear parameterization is sufficient to model the class of policies of the particular queueing system under consideration.

After parameterization of the control policy is chosen, various optimization methods can be applied to find the optimal value of the parameters with respect to the expected reward function. Usually such a method involves some version of a gradient ascent using estimates of the gradient of the reward function with respect to the parameters. One of the classical approaches is REINFORCE [Williams, 1992], which computes an estimator of the policy gradient using likelihood ratio [Aleksandrov et al., 1968, Glynn, 1987]. However, the likelihood ratio estimator is known to have high variance and REINFORCE estimator also suffers from that weakness. There have been other improvements to reduce the variance e.g. adding baseline terms or discarding some rewards in the GPOMDP estimator [Sutton and Barto, 2018, Zhao et al., 2011, Baxter and Bartlett, 2001, Pham et al., 2020] . Advanced policy optimization algorithms include Trust Region Policy Optimization (TRPO) [Schulman et al., 2015] and Proximal Policy Optimization [Schulman et al., 2017] which go beyond simple gradient accent with the aim to better control the steps and progress of the optimization algorithm.

The update step of the REINFORCE algorithm has the foundation from stochastic gradient descent (SGD) method [Robbins and Monro, 1951]. SGD with its stochastic first-order variants [Duchi et al., 2011, Kingma and Ba, 2014, Bottou et al., 2018, Nguyen et al., 2018] and variance reduction methods [Le Roux et al., 2012, Defazio et al., 2014, Johnson and Zhang, 2013, Nguyen et al., 2017] are favorable in machine learning because they are efficient in dealing with large-scale problems. In fact, other gradient estimators can be used to update the policy in place of the likelihood ratio method. Finite Difference estimators have been one of the popular choices, which is applied successfully to numerous applications in robotics [Peters and Schaal, 2006]. It has been observed that finite difference (FD) estimators generally have lower variance than the likelihood ratio estimator in typical REINFORCE algorithm [Peters and Schaal, 2006], which may result in a faster convergence of the optimization algorithm. However, such FD estimators are more computationally costly when the problem dimension is large. To properly compare both type of gradient estimators in application to queuing systems we embed them within an adaptive first order optimization method designed for noisy stochastic gradient and function estimators [Berahas et al., 2019, Jin et al., 2021]. This method is simpler than methods for policy gradients (e.g. TRPO and PPO [Schulman et al., 2015, 2017]) but is more advanced than those using FD

estimators. Applying this method for optimization when the optimal policy is known helps us gain knowledge about different gradient estimators and compare their performance.

Queueing systems control problem. We consider queueing networks and stochastic processing networks, where "job" arrive into the system and join a queue to be processed by a server [Prabhu, 1986, Perkins and Kumar, 2000, Srikant and Ying, 2013, Nguyen and Stolyar, 2016, Dai and Shi, 2019]. The control problem is defined as a process of allocating servers to processing jobs. In the queuing networks where interarrival times and service times are exponentially distributed, the control problem can be modeled as a Markov decision process (MDP) via uniformaization [Serfozo, 1979, Hazeghi and Puterman, 1995]. The objective function of interest is often the holding cost, where jobs in queue i incur a holding cost c_i for every time unit spent waiting for service [Dai and Gluzman, 2020, Van Mieghem, 1995, Mandelbaum and Stolyar, 2004, Krishnasamy et al., 2018]. The problem of minimizing the holding costs is equivalent to the maximization of rewards in a reinforcement learning framework, and we refer to these problems interchangeably in this paper. In the queueing networks literature, value-based approximate dynamic programming algorithms have been a popular approach [Chen and Yao, 1993, Chen et al., 2010, Veatch, 2010]. On the other hand, we aim to use a model-free policy gradient approach to learn the optimal policy directly. Policy gradient methods are favorable in many RL settings because they are effective and scalable to high dimension problems.

For a single server parallel queueing system, the so-called c-μ rule is known to minimize the expected holding cost by prioritizing serving the queues based on their holding cost c_i and service time μ_i [Van Mieghem, 1995, Mandelbaum and Stolyar, 2004] ratios. Although this rule is simple and easy to implement, it is not obvious to learn when the model parameters are unknown. In prior work [Krishnasamy et al., 2018] a greedy algorithm based on the max weight optimization problem was proposed to learn the c-μ for the single server parallel queueing system. In this paper, in contrast, we aim to learn the optimal solution directly from the RL environment. Although the stochastic networks are often far more complex than our simple setting, we believe that our motivations and methods are applicable to more complicated queueing systems, and to other RL applications in general.

2 Model description

2.1 Reinforcement Learning Model

Reinforcement learning problems are modeled as Markov Decision Processes $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \mu)$, where $\mathcal{S}$ is the state space and $\mathcal{A}$ is the action space [Agarwal et al., 2019, Sutton and Barto, 2018]. Initial state s_0 is chosen following some distribution μ. At each time step $t = 0, 1, 2, \ldots$, the agent takes an action $a_t \in \mathcal{A}$, obtains the immediate reward $r_t = r(s_t, a_t)$, and observes the next state s_{t+1}. $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is a transition function where $P(s' | s, a)$ denotes the probability of moving to entering state s' from s after taking action a. The state-action record up to time t:

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, ..., s_{t-1}, a_{t-1}, r_{t-1}, s_t) \tag{1}$$

is called a trajectory or history. The average cost of a sample path τ with horizon $T-1$ is $C(\tau) = \frac{1}{T}\sum_{t=0}^{T-1} c(s_t, a_t)$. Our goal is to find a differentiable parameterized policy function $\pi(A)$ that minimizes the expected long term average cost:

$$\min_A \left\{ J(A) := \mathbb{E}_{\tau\sim\pi(A)} C(\tau) = \mathbb{E}_{\tau\sim\pi(A)} \left[\lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} c(s_t, a_t) \right] \right\}, \tag{2}$$

where $C(\tau)$ is the respective average cost for trajectory τ.

2.2 Formulation of Parallel Queue System as a Reinforcement Learning Environment

We consider a parallel system with m queues. This model is simple and has been investigated thoroughly in classical queueing literature [Van Mieghem, 1995], thus, it is a good reference example to develop a better understanding of RL methods. We assume that the jobs of class i arrive to the system following Poisson processes with respective rates λ_i, $i = 1, \ldots, m$. There is one server which processes one job at a time. Let us assume that the processing times for class i jobs are i.i.d., having exponential distribution with the respective service rates $\mu_1, \mu_2, \ldots, \mu_m$ (thus the expected processing time for a job in class i is $\frac{1}{\mu_i}$. Our model is described in Figure 1. The corresponding load condition for this system is:

$$\rho = \frac{\lambda_1}{\mu_1} + \frac{\lambda_2}{\mu_2} + \cdots + \frac{\lambda_m}{\mu_m} < 1 \tag{3}$$

We assume that a decision time occurs when a new job arrives to the system or when the server completes a service. We also use a preemptive policy - when a decision occurs and the server is still in the middle of processing, then the

server preempts the unfinished class and follows the new action guided by the decision. Since the inter-arrival times and service times are exponentially distributed, this preemptive process does not alter this assumption.

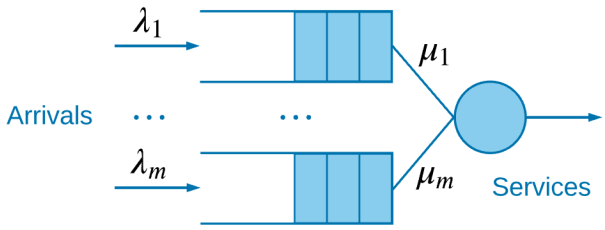

Figure 1: Simple parallel queueing system with one service unit.

The system state is captured in a vector $S = (s, p) \in \mathbb{N}^m \times \mathbb{N}$, where $s \in \mathbb{N}^m$ is the observable state that shows the number of jobs from each class $1, \ldots, m$ waiting in the system, and the last coordinate $p \in \{1, \ldots, m\}$ shows the server location p while it is processing a job of class p. While a queueing system may have infinite capacity in classic theoretical settings, it is more reasonable to consider a finite buffer in real-life applications. We will consider the latter case in our motivations and empirical findings, where every queue in the system has a capacity of N jobs, and any new arrival of a class which finds N jobs in that class will be lost (without affecting the cost).

Somewhat contrary to intuition, we allow the server to idle even when there is a job in the system, which implies that there is a total $m+1$ - actions $1 = 1, \ldots, m$ select job for class i for processing and the last action lets the server be idle. The server **always** chooses the next position based on the action provided by the policy and may choose to to serve the queue (class) with no job, or it may choose to remain idle even when there is a job in the system. A preemptive policy and an idle action allow us to model the problem more efficiently, and they have been common practices in queueing literature [Chang, 1965, White and Christie, 1958, Dai and Gluzman, 2020]. Since the state transition times are exponentially distributed, we can apply the uniformization technique to this problem [Serfozo, 1979, Hazeghi and Puterman, 1995]. For the queueing problem after uniformization, the new decision times are determined by the arrival times of a Poisson process with uniform rate that is independent of the current underlying state. Classical queueing theory characterizes the existence of a stationary Markovian policy when the load condition (3) is satisfied (e.g. [Meyn, 1998]).

We let $c \in \mathbb{R}^m$ be the holding cost vector, and let the holding cost be $c^\top s$ where $s \in \mathbb{R}^m$ is the observable state. In the RL setting, the corresponding reward function is $-c^\top s$. The following Theorem characterizes the optimal policy for this system.

Theorem 1. *For a parallel single server queueing system with infinite buffer, the optimal policy is the* priority policy *based on the c-μ rule: The server selects the job in queue $i^* = \text{argmax}\{c_i\mu_i | \text{for all } i \text{ such that } s_i > 0\}$ (i.e. choose the job in the class with the largest cost - expected processing time ratio among classes that have jobs waiting).*

This theorem is a classical result [Van Mieghem, 1995, Mandelbaum and Stolyar, 2004, Krishnasamy et al., 2018]. We denote the priority policy by π_P. It is important to note that we can only guarantee the optimality of π_P under the condition that the system has infinite length and the load condition is satisfied. When the capacity N is big enough, it is natural to expect that the system dynamic is not very different from the infinite buffer case and the priority policy still approximates the optimal solution. In the next section, we discuss the parameterization choices and how to represent the priority policy π_P in these settings.

3 New parameterization for queueing models

3.1 Parameterizations for Policy Optimization

We consider a simple class of linear parameterized policies that map from the state space to a distribution over the action space. Given the state vector $s \in \mathbb{N}^m$ a policy is defined matrix a $A \in \mathbb{R}^{(m+1)\times m}$ as follows

$$\pi(A) = \text{softmax}(z) = \text{softmax}(As), \tag{4}$$

where $z = As \in \mathbb{R}^{m+1}$ is the linear function of the observable state vector and the *softmax* function defined as

$$\text{softmax}(z) = \left(\frac{e^{z_1}}{\sum_{k=1}^{m+1} e^{z_k}}; \ldots; \frac{e^{z_{m+1}}}{\sum_{k=1}^{m+1} e^{z_k}} \right)^\top \in \mathbb{R}^{m+1}. \tag{5}$$

Under such a policy, the probability of any action for any state vector is a number between 0 and 1. In contrast, the priority policy π_P is a threshold policy, that is given any state vector the policy chooses one action with probability one. This policy cannot be represented as a differentiable function and thus is not amenable to gradient based algorithms. Our class policy is differentiable but does not contain π_P. In the next section we show that π_P can be approximated arbitrarily closely with our class of parameterized policies.

3.2 Representation of Priority Queue

Without loss of generality, let us assume that $c_1\mu_1 \geq c_2\mu_2 \geq \cdots \geq c_m\mu_m$. In other words, the deterministic priority policy π_P chooses to serve at queue i if and only if $i = \arg\min\{i : s_i > 0\}$ where $s \in \mathbb{N}^m$ is the current observable state vector. The following Theorem shows a trajectory that follows the priority policy.

Theorem 2. *Consider the call of linear policies described in Section 3.1 and assume that $s_i \leq Q$ for every $i = 1, \ldots, m$ for a given state vector. Let $\{A_k, k \geq 0\}$ be the sequence*

$$A_k = \begin{bmatrix} (Q+1)^m \cdot k + 1 & \ldots & 1 \\ \ldots & (Q+1)^i \cdot k + 1 & \ldots \\ 1 & \ldots & (Q+1) \cdot k + 1 \\ 1 & \ldots & 1 \end{bmatrix} \in \mathbb{R}^{(m+1)\times m}, \tag{6}$$

and $\pi_k = \pi(A_k) \in \mathbb{R}^{m+1}$ be the policy corresponding to the matrix A_k. We then have

- *The starting point of the sequence π_0 is a random policy that chooses every action with equal probability.*
- *The sequence π_k converges to the priority policy: $\pi_k \to \pi_P$ when $k \to \infty$.*
- *There does not exist a bounded sequence of matrices A'_k such that sequence $\pi(A'_k)$ converges to the priority policy: $\pi_k \to \pi_P$ when $k \to \infty$.*

This theorem suggests some interesting observations about the priority policy π_P and our linear policy class. It shows the existence of a trajectory which starts at a random policy and eventually approximates priority policy π_P arbitrarily closely. This implies that the linear class is sufficient to approximately learn the priority policy. Though there may be different trajectories in the parameter space that leads to the policy π_P, Theorem 2 shows that the in order to converge to the priority sequence, the policy parameters have to grow arbitrarily large, moreover, their growth may be as fast as exponential in the number of queues. We note that this result, with some possible variations, is applicable in many other settings, where optimal control policies are deterministic and can be similarly expressed as a limit of stochastic policy sequences parameterized by the softmax function.

Based on these observations, we consider an alternative parameterization for π. We let $B = \ln(A)$ where $\ln$ is an element-wise operator and A is a matrix with positive elements. Let B be the parameters of policy $\bar{\pi}$, we let the parameterization be

$$\bar{\pi}(B) = \text{softmax}(e^B s), \tag{7}$$

where the softmax function is defined above. We denote this scheme as the logarithm-scale parameterization of policy $\bar{\pi}$. It follows immediately that $\pi(A) = \bar{\pi}(B)$ for every matrix A, B satisfying that $A = e^B$. The sequence of matrices B_k associated with A_k defined in Theorem 2 is:

$$B_k = \begin{bmatrix} \ln[(Q+1)^m \cdot k + 1] & \ldots & 0 \\ \ldots & \ln[(Q+1)^i \cdot k + 1] & \ldots \\ 0 & \ldots & \ln[(Q+1) \cdot k + 1] \\ 0 & \ldots & 0 \end{bmatrix} \in \mathbb{R}^{(m+1)\times m}.$$

The diagonal elements B_k also grow infinitely large but much slower than those of A_k, which may result in better behavior of the optimization problem (e.g. smoother objective) when using this parameterization. The question arises if the logarithmically parametrized policy class is more restrictive than the linear class, because it generates only positive-element matrices. The following Theorem suggests that there is an equivalent positive matrix policy to every matrix A.

Theorem 3. *Let us consider the standard linear parameterization. For every matrix A, there exists a matrix A' that $\pi(A') = \pi(A)$ and $A' > 0$.*

4 Gradient Estimators for Policy Optimization

We now turn to describing the key ingredient in policy optimization - the function and gradient estimates. The function value estimator, which we call *inexact zeroth-order oracle* is average cost function induced by the parameterized policy $\pi(A)$. Let n be the number of sample paths and $T-1$ be the time horizon, we define the zeroth-order oracle $\tilde{J}(A)$ as

$$\tilde{J}(A) = \frac{1}{n}\sum_{i=1}^{n} C(\tau_i) = \frac{1}{n}\sum_{i=1}^{n}\left[\frac{1}{T}\sum_{t=0}^{T-1} c(s_{i,t}, a_{i,t})\right], \tag{8}$$

where τ_i is the i-th sample path following policy $\pi(A)$.

We now describe two popular ways of estimating gradients, thus providing *inexact first-order* oracles.

Finite Difference Estimator. The Finite Difference (FD) estimator $\widetilde{\nabla} J_1(A)$ has the form

$$\widetilde{\nabla} J_1(A) = \frac{1}{M}\sum_{i=1}^{M} \frac{\tilde{J}(A + u\cdot\vec{v_i}) - \tilde{J}(A)}{u}\vec{v_i}, \tag{9}$$

where $\vec{v_i}, i = 1, \ldots, M$ is a set of vectors and u is the finite difference step parameter. The essential of this gradient estimator comes from the difference between the (noisy) function values at a parameter A and the reference point $A + u\cdot\vec{v_i}$. In this paper, we choose the canonical setting where $\vec{v_i}, i = 1, \ldots, M$ are the unit vectors of the parameter space [Schlögl, 2018, Tu et al., 2019]. On the other hand, the standard Gaussian random vector is another popular choice for the increment direction [Balasubramanian and Ghadimi, 2018].

Policy Gradient Estimator. The classical REINFORCE (PG) method uses the likelihood ratio function to derive its gradient estimator [Aleksandrov et al., 1968, Glynn, 1987, Williams, 1992]. This estimator is based on the policy gradient theorem [Sutton et al., 1999] showing that $\nabla J(A) = \mathbb{E}_{\tau\sim\pi(A)}\left[\nabla \log p_A(\tau) R(\tau)\right]$, where p_A denotes the probability of trajectory τ under policy $\pi(A)$. However, the likelihood ratio estimator usually has high variance and REINFORCE estimator also suffers from that weakness. Variance reductions techniques have shown improvements to improve the efficiency of this estimator [Sutton and Barto, 2018, Zhao et al., 2011, Baxter and Bartlett, 2001, Kakade, 2001]. In this paper, we choose to present the common practice that adds a baseline term to the final estimator [Zhao et al., 2011, Peters and Schaal, 2006, Riedmiller et al., 2007]. For the time horizon T and number of sample paths N, our Policy Gradient (PG) estimator $\widetilde{\nabla} J_2(A)$ is computed as

$$\begin{aligned}
\widetilde{\nabla} J_2(A) &= \frac{1}{N}\sum_{i=1}^{N}\left[\frac{1}{T}\sum_{t=0}^{T-1}\nabla_A \log\pi(A)(s_{i,t}, a_{i,t})\right]\left[\frac{1}{T}\sum_{t=0}^{T-1} r(s_{i,t}, a_{i,t}) - b\right], \text{ where} \\
b &= \frac{\frac{1}{N}\sum_{i=1}^{N}\left[\frac{1}{T}\sum_{t=0}^{T-1}\nabla_A \log\pi(A)(s_{i,t}, a_{i,t})\right]^2\left[\frac{1}{T}\sum_{t=0}^{T-1} r(s_{i,t}, a_{i,t})\right]}{\frac{1}{N}\sum_{i=1}^{N}\left[\frac{1}{T}\sum_{t=0}^{T-1}\nabla_A \log\pi(A)(s_{i,t}, a_{i,t})\right]^2}.
\end{aligned} \tag{10}$$

Comparing two gradient estimators. In order to further understand the properties of our first-order oracles, we conduct a small experiment to investigate their behavior. Both of these gradient estimators have been popular in the literature, however, as far as we know they have not been compared side by side. The main reason for this is that they are typically implemented within algorithms customized the the specific choice of first order oracle. In our experiment, we choose the number of samples n and N such that two estimators have the same sample complexity. We compute two estimators at two reference points: at a random policy and at the (optimal) priority policy. We evaluate the covariance matrices of these stochastic estimators and plot the results in Figure 2.

This experiment shows that the REINFORCE estimator has similar or slightly higher variance than Finite Difference estimator at the random policy. When we compute the two estimators at the (optimal) priority policy, the FD estimator has an extremely low variance and outperforms the PG estimator. Thus, we conclude that the FD estimator is more reliable than PG estimator when the algorithm reaches the neighborhood of minimizers.

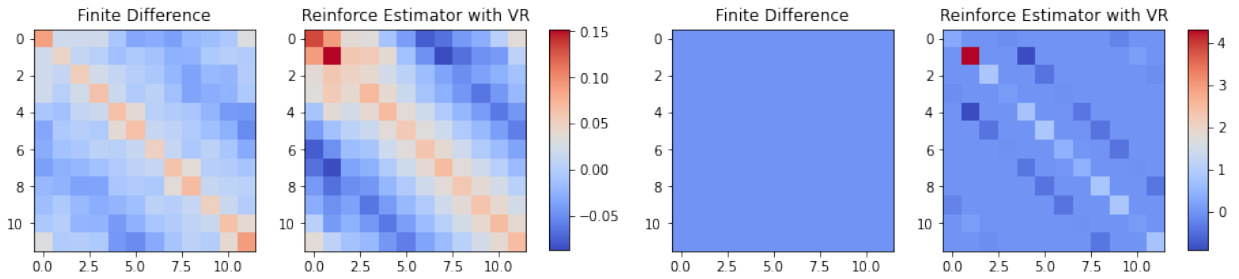

Figure 2: Comparisons of the covariance matrices of Finite Difference (FD) estimator and REINFORCE (PG) estimator with Variance Reduction, computed at random policy (left) and (optimal) priority policy (right). For the random policy, PG estimator behaves in the relatively same scale as FD estimator. However, FD is still significantly better than PG estimator with variance reduction at the priority policy.

5 Numerical Experiments

In this section, we experiment with two parameterizations: the standard linear parameterization and the logarithm-scale parameterization proposed in Section 3.2. We implement two gradient estimators and employ an adaptive line-search algorithm (ALOE) to optimize the cost function. Unlike other more complex policy optimization algorithms (e.g. TRPO and PPO [Schulman et al., 2015, 2017]) this method, describe it in Algorithm 1, has theoretical complexity guarantees for both types on first order oracles we describe above and allows for bias in zeroth-order and first-order oracles [Jin et al., 2021, Berahas et al., 2019].

Algorithm 1 Adaptive Line-search with Oracle Estimations (ALOE) [Berahas et al., 2019]

Input: Parameter ϵ_f of the zeroth-order oracle, starting point A_0, max step size $\alpha_{\max} > 0$, initial step size $\alpha_0 < \alpha_{\max}$, constants $\theta, \gamma \in (0, 1)$.
for $k = 0, 1, 2, \cdots$ **do**
  Compute gradient approximation $g_k = \widetilde{\nabla} J(A_k)$
  Check sufficient decrease:
    Let $A_k^+ = A_k - \alpha_k g_k$. Generate the zeroth-order oracle $\tilde{J}(A_k)$ and $\tilde{J}(A_k^+)$.
    Check the modified Armijo condition:

$$\tilde{J}(A_k^+) \leq \tilde{J}(A_k) - \alpha_k \theta \|g_k\|^2 + 2\epsilon_f. \tag{11}$$

  Successful step: If (11) holds, then set $A_{k+1} = A_k^+$ and $\alpha_{k+1} = \min\{\alpha_{\max}, \gamma^{-1}\alpha_k\}$.
  Unsuccessful step: Otherwise, set $A_{k+1} = A_k$ and $\alpha_{k+1} = \gamma\alpha_k$.
end for

Compare the cost function for two parameterizations. We experiment with two parameterizations of the policy with various load conditions of the system from light to heavy traffic. We summarize our setting in Table 1 and highlight the best results obtained by our algorithms there. In addition, we plot the training progress in Figure 3. In each setting, we tune the algorithms using grid search and pick the best choice of hyper-parameters (e.g. the step sizes $\alpha_{\max}$ and α_0) to the training stage. We repeat all the stochastic experiments for 5 random seeds, then report the average performance. We present the detailed implementation settings in the Appendix.

Table 1 shows that in most cases, the logarithm-scales perform better than the standard linear-scales. The Finite Difference estimator often shows better performance than Policy Gradient estimator, from medium to heavy load conditions. We also report the confidence interval (CI) of the cost function evaluated at the optimal (priority) policy. Since ALOE accepts steps based on a modified function reduction condition, the obtained cost values tend to be closer to the lower end of the corresponding confidence intervals.

Table 1: Summary of the load conditions in our experiment setting, and the best performance of each parameterization (Linear scale and Logarithm scale). The load parameter ρ is presented in equation (3). The cost function is normalized to $[0, 1]$. We report the best cost function achieved in each setting and note the first order oracle that yields the best result. The last row reports the confidence intervals of the cost function at the optimal (priority) policy in each setting.

Settings	Low Load	Medium Load	Balanced Load	Heavy Load
Service rates	$(18, 9, 6)$	$(9, 4.5, 3)$	$(6, 3, 2)$	$(3, 2, 1)$
Load parameter ρ	$1/3$	$2/3$	1	$11/6$
Linear scale	0.0003 (PG)	0.0018 (FD)	0.0097 (FD)	0.2647 (FD)
Logarithm scale	0.0004 (PG)	0.0016 (FD)	0.0066 (FD)	0.2586 (FD)
Optimal cost CI	$[0.0002, 0.0015]$	$[0.0012, 0.0032]$	$[0.0079, 0.0093]$	$[0.2611, 0.4037]$

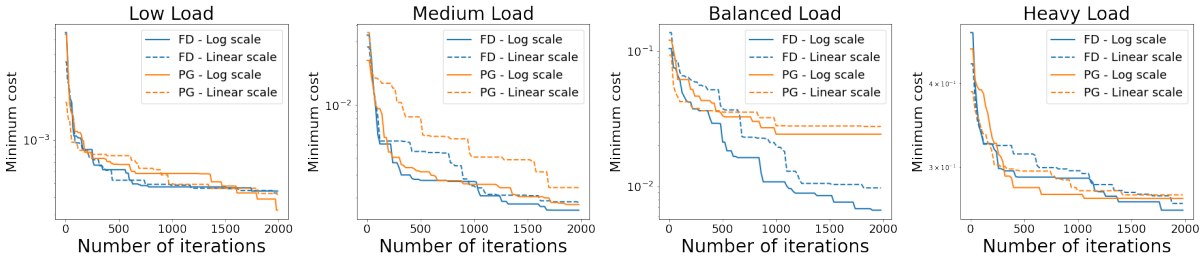

Figure 3: Comparisons of the minimum cost function achieved by iteration, for two parameterization scales: logarithm (proposed scale, presented in continuous solid lines) and linear scale (presented in dashed line). We show the performance of Finite Difference (FD) estimator and REINFORCE estimator with variance reduction (PG) in different colors.

Compare the learned policy with the priority policy. Now we compare the learned policies with the priority policy and show how they behave relatively to the desired outcome. For every pair of state and action induced by policy $\pi(A)$, we compute the rate that $\pi(A)$ chooses the "correct" action dictated by priority policy π_P. We keep measuring that correct rate along the training process, and report the results in Figure 4.

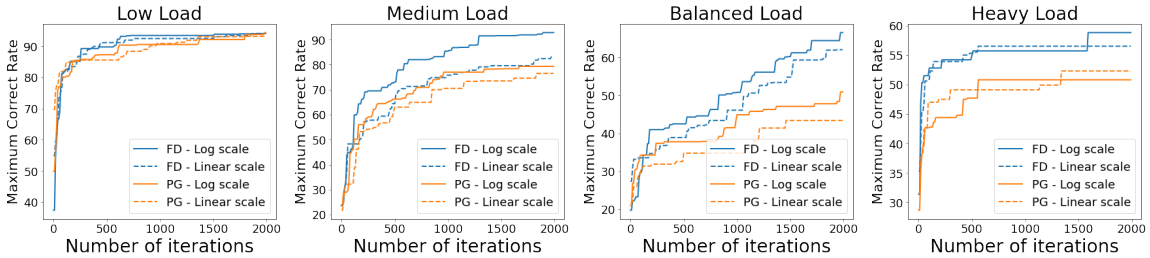

Figure 4: Comparisons of the maximum correct rate (in percent) achieved by iteration for two parameterization scales: logarithm (proposed scale, presented in continuous solid lines) and linear scale (presented in dashed line). We show the performance of Finite Difference (FD) estimator and REINFORCE estimator with variance reduction (PG) in different colors.

Our summary in Table 2 shows that the logarithm scale usually has a better performance than the normal linear parameterization. In addition, it suggests that the algorithms learn the correct action more easily in the low and medium load conditions, with higher than 90 percent accuracy rates. In the balanced and heavy load settings, it is more difficult for the algorithms to learn the correct actions, however, it still achieves an accuracy rate of approximately 60 percent. We present the experiment setting in the Appendix.

Table 2: The best correct rate in each setting achieved by our algorithm, with the corresponding gradient estimators that yields the best reported result.

Settings	Low Load	Medium Load	Balanced Load	Heavy Load
Service rates	$(18, 9, 6)$	$(9, 4.5, 3)$	$(6, 3, 2)$	$(3, 2, 1)$
Load parameter ρ	$1/3$	$2/3$	1	$11/6$
Linear scale	94.3 (FD)	83.2 (FD)	62.0 (FD)	56.5 (FD)
Logarithm scale	94.4 (PG)	92.8 (FD)	66.5 (FD)	58.8 (FD)

6 Conclusion

We investigate the optimization aspect of queueing reinforcement learning environments, by proposing new logarithm-scale parameterization and also comparing two gradient estimators side-by side within a stochastic step search algorithm. In most settings, the logarithm-scale parameterization and the finite difference based gradient estimator show better performance than the linear parametrization and policy gradient estimators.

Acknowledgements

The authors sincerely thank Jamol J. Pender and Shane G. Henderson for their insightful discussions and useful suggestions to complete this project. The work of Trang H. Tran and Katya Scheinberg have partly been supported by the ONR Grant N00014-22-1-2154.

APPENDIX

A Technical Proofs

Proof of Theorem 2

Proof. From the proof of Theorem 3, we note that it is sufficient to prove the statements for the following sequence:

$$A''_k = \begin{bmatrix} (Q+1)^m \cdot k & \dots & 0 \\ \dots & (Q+1)^i \cdot k & \dots \\ 0 & \dots & (Q+1) \cdot k \\ 0 & \dots & 0 \end{bmatrix} \in \mathbb{R}^{(m+1)\times m},$$

because $A_k = A''_k + 1$ and $\pi(A_k) = \pi(A''_k)$.

The first statement of Theorem 2 when $k = 0$ is straightforward. We note that $A''_0 = 0$ and the softmax function at zero vector returns a random policy that chooses every action with equal probability.

Now we move to the second statement. Let $z_k = A''_k s$, we have:

$$\begin{aligned} z_k &= \left((Q+1)^m \cdot ks_1; \dots; (Q+1)^i \cdot ks_{m-i}; \dots; (Q+1) \cdot ks_m; 0\right) \\ &= k\left((Q+1)^m s_1; \dots; (Q+1)^i s_{m-i}; \dots; (Q+1)s_m; 0\right) = k \cdot y, \end{aligned}$$

where s_i is the i-th element of the observable state s, and

$$y = \left((Q+1)^m s_1; \dots; (Q+1)^i s_{m-i}; \dots; (Q+1)s_m; 0\right) \in \mathbb{R}^{m+1},$$

Now let $j = \arg\min\{j : s_j > 0\}$ and assuming that j is not an idle action, we have that the deterministic priority policy π_P chooses to serve at queue j. We prove that y_j is the largest element of vector $y \in \mathbb{R}^{m+1}$.

Firstly, we have $s_i = 0$ for every $i < j$ and therefore $y_i = 0$ for every $i < j$ and $y_j = (Q+1)^{m-j} s_j > 0 = y_i$ for every $i < j$. In addition, since $s_i \le Q$ for every $i > j$, we have that $s_i < Q + 1$ and

$$y_i = (Q+1)^{m-i} s_i < (Q+1)^{m-i+1} \le (Q+1)^{m-j} \le (Q+1)^{m-j} s_j = y_j.$$

Hence y_j is the largest element of vector $y \in \mathbb{R}^{m+1}$. Now we consider the softmax policy induced by $z_k = k \cdot y$:

$$\text{softmax}(k \cdot y) = \left(\frac{e^{ky_1}}{\sum_{i=1}^{m+1} e^{ky_i}}; \dots; \frac{e^{ky_{m+1}}}{\sum_{i=1}^{m+1} e^{ky_i}} \right)^\top \in \mathbb{R}^{m+1}.$$

Note that ky_j is the largest element of vector $ky \in \mathbb{R}^{m+1}$. Hence when k goes to infinity, the vector softmax$(k \cdot y)$ converges to the j-th unit vector in $\mathbb{R}^{m+1}$. This is the deterministic priority policy π_P chooses to serve at queue j.

Now we move to the final statement. Let us fixed the state s we have that $(\pi_P)_i = 0$ for every incorrect action, i.e. action $i \neq j$. We assume the contradiction that there exist a sequence A'_k such that all the (absolute value) of elements of A'_k are upper bounded by a constant R, and sequence $\pi(A'_k)$ converges to the priority policy: $\pi_k \to \pi_P$ when $k \to \infty$.

It is easily seen that all the (absolute value) of elements of the vector $z'_k = A'_k s$ are upper bounded by mRQ, since the state space is bounded by Q. Thus the denominator of the softmax vector (i.e. sum of vector $\exp(z'_k)$) is upper bounded by $(m+1)e^{mRQ}$. However, for an action $i \neq j$, we have the softmax value converge to 0. This shows that $\exp((z'_k)_i)$ converges to 0 and $(z'_k)_i \to -\infty$, which contradicts to the fact that z'_k and A'_k are upper bounded.

Hence we complete the proof. Note that the construction of sequence A_k is not unique, and there may be different trajectories in the parameter space that leads to the policy π_P. However, this result suggests that the limit of such trajectory need to have the ability to distinguish (and let dominate) the more important queue. Such ability leads to the big elements of A_k when k is large. □

Proof of Theorem 3

Let us consider the standard linear parameterization. For every matrix A, there exists a shifted matrix A' that $\pi(A') = \pi(A)$ and $A' > 0$.

Proof. We recall that each policy is represented by a matrix $A \in \mathbb{R}^{(m+1)\times m}$, and the output distribution is

$$\pi(A) = \text{softmax}(z) = \text{softmax}(As),$$

where $z = As \in \mathbb{R}^{m+1}$ is the linear function output. The *softmax* function is defined as

$$\text{softmax}(z) = \left(\frac{e^{z_1}}{\sum_{k=1}^{m+1} e^{z_k}}; \ldots; \frac{e^{z_{m+1}}}{\sum_{k=1}^{m+1} e^{z_k}} \right)^\top \in \mathbb{R}^{m+1}.$$

We prove that for every number $\mu \in \mathbb{R}$, the following statement holds: $\pi(A) = \pi(A + \mu)$, where $A + \mu$ yields the element-wise addition between the elements of A and μ. In fact, we have

$$\begin{aligned}
\pi(A + \mu) &= \text{softmax}((A + \mu)s) \\
&= \text{softmax}\left(As + \vec{e}_{m+1} \cdot \mu \sum_{k=1}^{m+1} s_k \right) \\
&= \text{softmax}\left(As \right) = \pi(A),
\end{aligned}$$

where $\vec{e}_{m+1}$ is the vector containing all elements 1 in $\mathbb{R}^{m+1}$. The last line follows since the softmax function is invariant with respect to the addition of the scalar of $\vec{e}_{m+1}$.

Finally, choosing μ such that all the elements of $A + \mu$ is postive, we have $\mu = 1 - \min A_{ij}$ and $A' = A + \mu$ satisfying that $\pi(A') = \pi(A)$ and $A' > 0$. □

B Experiment Settings

We describe the detailed implementation settings for our experiments here.

We model the queueing system dynamics using OpenAI gym package Brockman et al. [2016]. We implement the zeroth- and first-order oracles for our environment using a warm up stage of 1000 iterations to reduce the initial variance. In addition, the cost function is normalized to $[0, 1]$. In more detail, the cost function is $1/(mQ)$ where m is the number of queues in the system and Q is the maximum number of jobs in every queue. Thus, the cost function is 0 when the system has no job, and it is 1 when the system is completely full with mQ jobs.

The service rates for each setting is given in Table 1. We choose $m = 3$ and $Q = 100$. We choose the number of sample paths n and N such that two estimators have the same sample complexity. The number of sample paths n in the zero-th order oracle is 20. The number of (basis) vector used in the FD estimator is the size of A, which is $m(m + 1)$. Hence we choose the number of sample paths of PG estimator to be $N = [m(m + 1) + 1]n$ to match the number of sample paths of FD estimator.

The time horizon T is 100 and we optimize over 2000 iterations of ALOE algorithm. We start all the algorithms at random policy $A = 0$. We tune different hyper-parameters for ALOE algorithm using grid search and choose the best parameters in to the final stage. The final setting of ALOE algorithm is: $\epsilon_f = 0.01$, $\alpha_{\max} = 2, \alpha_0 = 0.01$, $\theta = 0.01$ and $\gamma = 0.5$. In Table 1, we compute the confidence intervals using 1000 sample paths and with horizon 100. We also use a warm up stage with 1000 iterations.

References

Alekh Agarwal, Nan Jiang, Sham M. Kakade, and Wen Sun. Reinforcement learning: Theory and algorithms, 2019. URL https://rltheorybook.github.io/.

V.M. Aleksandrov, V.I. Sysoyev, and V.V. Shemeneva. Stochastic optimization. *Engineering Cybernetics*, 5:11–16, 01 1968.

Krishnakumar Balasubramanian and Saeed Ghadimi. Zeroth-order (non)-convex stochastic optimization via conditional gradient and gradient updates. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/36d7534290610d9b7e9abed244dd2f28-Paper.pdf.

J. Baxter and P. L. Bartlett. Infinite-horizon policy-gradient estimation. *J. Artif. Int. Res.*, 15(1):319–350, November 2001.

Albert Berahas, Liyuan Cao, and Katya Scheinberg. Global convergence rate analysis of a generic line search algorithm with noise. *SIAM Journal on Optimization*, 10 2019.

L. Bottou, F. E. Curtis, and J. Nocedal. Optimization Methods for Large-Scale Machine Learning. *SIAM Rev.*, 60(2): 223–311, 2018.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

Wei Chang. Preemptive priority queues. *Operations Research*, 13(5):820–827, 1965. ISSN 0030364X, 15265463. URL http://www.jstor.org/stable/167731.

Hong Chen and David Yao. Dynamic scheduling of a multiclass fluid network. *Operations Research*, 41:1104–1115, 12 1993. doi: 10.1287/opre.41.6.1104.

Wei Chen, Dayu Huang, Ankur Kulkarni, Jayakrishnan Unnikrishnan, Quanyan Zhu, Prashant Mehta, Sean Meyn, and Adam Wierman. Approximate dynamic programming using fluid and diffusion approximations with applications to power management. *Proceedings of the 48th IEEE Conference on Decision and Control*, pages 3575 – 3580, 01 2010. doi: 10.1109/CDC.2009.5399685.

J. Dai and Pengyi Shi. Inpatient overflow: An approximate dynamic programming approach. *Manufacturing and Service Operations Management*, 21, 05 2019. doi: 10.1287/msom.2018.0730.

J. G. Dai and Mark Gluzman. Queueing network controls via deep reinforcement learning, 2020.

Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in Neural Information Processing Systems*, pages 1646–1654, 2014.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

Peter Glynn. Likelilood ratio gradient estimation. *Winter Simulation Conference Proceedings*, pages 366–375, 12 1987. doi: 10.1145/318371.318612.

H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 2094–2100, 2016.

Kasra Hazeghi and Martin Puterman. Markov decision processes: Discrete stochastic dynamic programming. *Journal of the American Statistical Association*, 90:392, 03 1995. doi: 10.2307/2291177.

Billy Jin, Katya Scheinberg, and Miaolan Xie. High probability complexity bounds for line search based on stochastic oracles. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 9193–9203. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/file/4cb811134b9d39fc3104bd06ce75abad-Paper.pdf.

Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *NIPS*, pages 315–323, 2013.

S. Kakade. A natural policy gradient. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, pages 1531–1538, Cambridge, MA, USA, 2001. MIT Press.

D. P. Kingma and J. Ba. ADAM: A Method for Stochastic Optimization. *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, abs/1412.6980, 2014.

Subhashini Krishnasamy, Ari Arapostathis, Ramesh Johari, and Sanjay Shakkottai. On learning the c-μ rule in single and parallel server networks, 10 2018.

Nicolas Le Roux, Mark Schmidt, and Francis Bach. A stochastic gradient method with an exponential convergence rate for finite training sets. In *NIPS*, pages 2663–2671, 2012.

Bai Liu, Qiaomin Xie, and Eytan Modiano. Reinforcement learning for optimal control of queueing systems. In *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 663–670, 2019. doi: 10.1109/ALLERTON.2019.8919665.

Avishai Mandelbaum and Alexander Stolyar. Scheduling flexible servers with convex delay costs: Heavy-traffic optimality of the generalized c-μ-rule. *Operations Research*, 52:836–855, 12 2004. doi: 10.1287/opre.1040.0152.

Sean Meyn. Stability and optimization of queueing networks and their fluid models, 09 1998.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing atari with deep reinforcement learning. *ArXiv*, abs/1312.5602, 2013.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

Lam Nguyen, Phuong Ha Nguyen, Marten van Dijk, Peter Richtarik, Katya Scheinberg, and Martin Takac. SGD and Hogwild! convergence without the bounded gradients assumption. In *Proceedings of the 35th International Conference on Machine Learning-Volume 80*, pages 3747–3755, 2018.

Lam M. Nguyen and Alexander L. Stolyar. A service system with randomly behaving on-demand agents. *SIGMETRICS Perform. Eval. Rev.*, 44(1):365–366, June 2016. ISSN 0163-5999. doi: 10.1145/2964791.2901484. URL http://doi.acm.org/10.1145/2964791.2901484.

Lam M. Nguyen and Alexander L. Stolyar. A queueing system with on-demand servers: local stability of fluid limits. *Queueing Systems*, Nov 2017. ISSN 1572-9443. doi: 10.1007/s11134-017-9564-8. URL https://doi.org/10.1007/s11134-017-9564-8.

Lam M Nguyen, Jie Liu, Katya Scheinberg, and Martin Takáč. SARAH: A novel method for machine learning problems using stochastic recursive gradient. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2613–2621. JMLR. org, 2017.

OpenAI. OpenAI Five. https://blog.openai.com/openai-five/, 2018.

Elena Pashenkova, Irina Rish, and Rina Dechter. Value iteration and policy iteration algorithms for markov decision problem, 1996.

James Perkins and Panganamala Kumar. Stable distributed, real-time scheduling of flexible manufacturing / assembly / disassembly systems, 07 2000.

Jan Peters and Stefan Schaal. Policy gradient methods for robotics, 11 2006.

Nhan Pham, Lam Nguyen, Dzung Phan, Phuong Ha Nguyen, Marten van Dijk, and Quoc Tran-Dinh. A hybrid stochastic policy gradient algorithm for reinforcement learning. In Silvia Chiappa and Roberto Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 374–385. PMLR, 26–28 Aug 2020. URL https://proceedings.mlr.press/v108/pham20a.html.

N. Prabhu. Re-entrant lines. *Queueing Systems - Theory and Applications - QUESTA*, 1:1–4, 06 1986. doi: 10.1007/BF01149325.

Martin Riedmiller, Jan Peters, and Stefan Schaal. Evaluation of policy gradient methods and variants on the cart-pole benchmark, 05 2007.

H. Robbins and S. Monro. A stochastic approximation method. *Ann. Math. Statist.*, 22(3):400–407, 09 1951.

Erik Schlögl. Finite difference methods for partial differential equations, 09 2018.

J. Schulman, S. Levine, P. Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pages 1889–1897, Lille, France, 07–09 Jul 2015.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017.

Richard Serfozo. Technical note–an equivalence between continuous and discrete time markov decision processes. *Operations Research*, 27:616–620, 06 1979. doi: 10.1287/opre.27.3.616.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. v. d. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

R. Srikant and Lei Ying. Communication networks: An optimization, control, and stochastic networks perspective, 11 2013.

R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning, 2nd Edition*. MIT Press, 2018.

R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, pages 1057–1063, 1999.

Chun-Chen Tu, Paishun Ting, Pin-Yu Chen, Sijia Liu, Huan Zhang, Jinfeng Yi, Cho-Jui Hsieh, and Shin-Ming Cheng. Autozoom: Autoencoder-based zeroth order optimization method for attacking black-box neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:742–749, 07 2019. doi: 10.1609/aaai.v33i01.3301742.

Jan Van Mieghem. Dynamic scheduling with convex delay costs: The generalized c-μ rule. *The Annals of Applied Probability*, 08 1995. doi: 10.1214/aoap/1177004706.

Michael Veatch. Approximate linear programming for networks: Average cost bounds. *Computers and Operations Research*, 63, 04 2010. doi: 10.1016/j.cor.2015.04.014.

C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.

Harrison White and Lee Christie. Queuing with preemptive priorities or with breakdown. *Operations Research*, 6: 79–95, 02 1958. doi: 10.1287/opre.6.1.79.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, May 1992.

T. Zhao, H. Hachiya, G. Niu, and M. Sugiyama. Analysis and improvement of policy gradient estimation, 2011.