A VIEW ON DEEP REINFORCEMENT LEARNING IN SYSTEM OPTIMIZATION

ABSTRACT

Many real-world systems problems require reasoning about the long term consequences of actions taken to configure and manage the system. These problems with delayed and often sequentially aggregated reward, are often inherently reinforcement learning problems and present the opportunity to leverage the recent substantial advances in deep reinforcement learning. However, in some cases, it is not clear why deep reinforcement learning is a good fit for the problem. Sometimes, it does not perform better than the state-of-the-art solutions. And in other cases, random search or greedy algorithms could outperform deep reinforcement learning. In this paper, we review, discuss, and evaluate the recent trends of using deep reinforcement learning in system optimization. We propose a set of essential metrics to guide future works in evaluating the efficacy of using deep reinforcement learning in system optimization. Our evaluation includes challenges, the types of problems, their formulation in the deep reinforcement learning setting, embedding, the model used, efficiency, and robustness. We conclude with a discussion on open challenges and potential directions for pushing further the integration of reinforcement learning in system optimization.

许多现实世界中的系统问题需要推理为配置和管理系统而采取的行动的长期后果。这些延迟的问题，往往是顺序汇总的奖励，往往是固有的强化学习问题，并提供了利用深度强化学习最近取得的实质性进展的机会。然而，在某些情况下，不清楚为什么深度强化学习很适合这个问题。有时，它的性能并不比最先进的解决方案更好。在其他情况下，随机搜索或贪婪算法可能优于深度强化学习。在本文中，我们回顾、讨论和评估了在系统优化中使用深度强化学习的最新趋势。我们提出了一套基本指标来指导未来评估在系统优化中使用深度强化学习的有效性。我们的评估包括挑战、问题类型、它们在深度强化学习环境中的表述、嵌入、使用的模型、效率和健壮性。最后，我们讨论了开放的挑战和进一步推动强化学习在系统优化中的整合的潜在方向。

1 INTRODUCTION

Reinforcement learning (RL) is a class of learning problems framed in the context of planning on a Markov Decision Process (MDP) (Bellman, 1957), when the MDP is not known. In RL, an agent continually interacts with the environment (Kaelbling et al., 1996; Sutton et al., 2018). In particular, the agent observes the state of the environment, and based on this observation takes an action. The goal of the RL agent is then to compute a policy–a mapping between the environment states and actions–that maximizes a long term reward. There are multiple ways to extrapolate the policy. Non-approximation methods usually fail to predict good actions in states that were not visited in the past, and require storing all the action-reward pairs for every visited state, a task that incurs a huge memory overhead and complex computation. Instead, approximation methods have been proposed. Among the most successful ones is using a neural network in conjunction with RL, also known as deep RL. Deep models allow RL algorithms to solve complex problems in an end-to-end fashion, handle unstructured environments, learn complex functions, or predict actions in states that have not been visited in the past. Deep RL is gaining wide interest recently due to its success in robotics, Atari games, and superhuman capabilities (Mnih et al., 2013; Doya, 2000; Kober et al., 2013; Peters et al., 2003). Deep RL was the key technique behind defeating the human European champion in the game of Go, which has long been viewed as the most challenging of classic games for artificial intelligence (Silver et al., 2016).

强化学习 (RL) 是一类在马氏决策过程 (MDP) (Bellman，1957) 规划的背景下构建的学习问题，当 MDP 未知时。在 RL 中，代理不断与环境相互作用 (kaelling 等人，1996; 萨顿等人，2018)。特别是，代理观察环境的状态，并根据这一观察采取行动。RL 代理的目标是计算一个策略 -- 环境状态和操作之间的映射 -- 最大限度地提高长期回报。有多种方法可以推断政策。非近似方法通常无法预测过去没有访问过的状态中的良好行为，并且需要存储每个访问过的状态的所有动作奖励对, 产生巨大内存开销和复杂计算的任务。相反，已经提出了近似方法。最成功的方法之一是将神经网络与 RL 结合使用，也称为 deep RL。深度模型允许 RL 算法以端到端的方式解决复杂的问题，处理非结构化环境，学习复杂的函数，或者预测过去从未访问过的状态中的动作。Deep RL 最近因其在机器人、 Atari 游戏和超人能力方面的成功而受到广泛关注 (Mnih 等人，2013; Doya，2000; Kober 等人，2013; 彼得斯等人，2003)。Deep RL 是在围棋游戏中击败欧洲人类冠军的关键技术，围棋一直被认为是人工智能最具挑战性的经典游戏 (Silver 等人, 2016)。

Many system optimization problems have a nature of delayed, sparse, aggregated or sequential rewards, where improving the long term sum of rewards is more important than a single immediate reward. For example, an RL environment can be a computer cluster. The state could be defined as a combination of the current resource utilization, available resources, time of the day, duration of jobs waiting to run, etc. The action could be to determine on which resources to schedule each job. The reward could be the total revenue, jobs served in a time window, wait time, energy efficiency, etc., depending on the objective. In this example, if the objective is to minimize the waiting time of all jobs, then a good solution must interact with the computer cluster and monitor the overall wait time of the jobs to determine good schedules. This behavior is inherent in RL. The RL agent has the advantage of not requiring expert labels or knowledge and instead the ability to learn directly from its own interaction with the world. RL can also learn sophisticated system characteristics that a straightforward solution like first come first served allocation scheme cannot. For instance, it could be better to put earlier long-running arrivals on hold if a shorter job requiring fewer resources is expected shortly.

许多系统优化问题具有延迟、稀疏、聚合或顺序奖励的性质，其中提高长期奖励的总和比单个即时奖励更重要。例如，RL 环境可以是计算机集群。状态可以定义为当前资源利用率、可用资源、一天中的时间、等待运行的作业持续时间等的组合。操作可以是确定安排每个作业的资源。奖励可以是总收入、在时间窗口服务的工作、等待时间、能源效率等。，取决于目标。在这个例子中，如果目标是最小化所有工作的等待时间, 然后，一个好的解决方案必须与计算机集群交互，并监控作业的整体等待时间，以确定好的时间表。这种行为是 RL 固有的。RL 代理的优点是不需要专家标签或知识，而是能够直接从自己与世界的互动中学习。RL 还可以学习复杂的系统特性，这是像先到先得服务分配方案这样简单的解决方案所不能的。例如，如果预计不久会有一个需要更少资源的较短工作，最好暂停提前的长期到达。

In this paper, we review different attempts to overcome system optimization challenges with the use of deep RL. Unlike previous reviews (Hameed et al., 2016; Mahdavinejad et al., 2018; Krauter et al., 2002; Wang et al., 2018; Ashouri et al., 2018; Luong et al., 2019) that focus on machine learning methods without discussing deep RL models or applying them beyond a specific system problem, we focus on deep RL in system optimization in general. From reviewing prior work, it is evident that standardized metrics for assessing deep RL solutions in system optimization problems are lacking. We thus propose quintessential metrics to guide future work in evaluating the use of deep RL in system optimization. We also discuss and address multiple challenges that faced when integrating deep RL into systems.

在本文中，我们回顾了使用 deep RL 克服系统优化挑战的不同尝试。与以前的评论不同 (Hameed 等人，2016; Mahdavinejad 等人，2018; Krauter 等人，2002; 王等人，2018; Ashouri 等人，2018; Luong 等人，2019) 专注于机器学习方法，而不讨论深层 RL 模型或将它们应用于特定的系统问题之外,我们在系统优化方面的重点是 deep RL。从回顾之前的工作来看，很明显，缺乏评估系统优化问题中的 deep RL 解决方案的标准化指标。因此，我们提出了典型的指标来指导未来评估 deep RL 在系统优化中的使用。我们还讨论并解决了将 deep RL 集成到系统中时面临的多重挑战。

2.1 Prior RL Works With Alternative Approximation Methods

Multiple prior works have proposed to use non-deep neural network approximation methods for RL in system optimization. These works include reliability and monitoring (Das et al., 2014; Zhu et al., 2007; Zeppenfeld et al., 2008), memory management (Ipek et al., 2008; Andreasson et al., 2002; Peled et al., 2015; Diegues et al., 2014) in multicore systems, congestion control (Li et al., 2016; Silva et al., 2016), packet routing (Choi et al., 1996; Littman et al., 2013; Boyan et al., 1994), algorithm selection (Lagoudakis et al., 2000), cloud caching (Sadeghi et al., 2017), energy efficiency (Farahnakian et al., 2014) and performance (Peng et al., 2015; Jamshidi et al., 2015; Barrett et al., 2013; Arabnejad et al., 2017; Mostafavi et al., 2018). Instead of using a neural network to approximate the policy, these works used tables, linear approximations, and other approximation methods to train and represent the policy. Tables were generally used to store the Q-values, i.e., one value for each action, state pair, which are used in training, and this table becomes the ultimate policy. In general, deep neural networks allowed for more complex forms of policies and Q functions (Lin, 1993), and can better approximate good actions in new states.

多个先前的工作已经提出在系统优化中使用非深度神经网络近似方法来实现 RL。这些工作包括可靠性和监控 (Das 等人，2014; 朱等人，2007; Zeppenfeld 等人，2008)，内存管理 (Ipek 等人，2008; Andreasson 等人, 2002; Peled 等人，2015; Diegues 等人，2014) 在多核系统中，拥塞控制莉等人，2016; 席尔瓦等人，2016)，分组路由 (崔等人，1996; Littman 等人，2013; Boyan 等人，1994), 算法选择 (lagoudais 等人，2000) 、云缓存 (Sadeghi 等人，2017) 、能效 (Farahnakian 等人,2014) 和性能 (彭等人，2015; Jamshidi 等人，2015; Barrett 等人，2013; 阿拉贝纳贾德等人，2017; Mostafavi 等人，2018)。这些工作使用表格、线性近似和其他近似方法来训练和表示策略，而不是使用神经网络来近似策略。表通常用于存储 q值，即每个操作的一个值，状态对，用于训练，此表成为最终策略。一般来说，深度神经网络允许更复杂的策略和 Q 函数形式 (林，1993)，并且可以更好地近似新状态下的良好行为。

3 RL IN SYSTEM OPTIMIZATION

In this section, we discuss the different system challenges tackled using RL and divide them into two categories: Episodic Tasks, in which the agent-environment interaction naturally breaks down into a sequence of separate terminating episodes, and Continuing Tasks, in which it does not. For example, when optimizing resources in the cloud, the jobs arrive continuously and there is not a clear termination state. But when optimizing the order of SQL joins, the query has a finite number of joins, and thus after enough steps the agent arrives at a terminating state.

在本节中，我们讨论了使用 RL 解决的不同系统挑战，并将它们分为两类: 情节任务, 其中，代理-环境交互自然分解为一系列单独的终止集和持续任务，其中它没有。例如，在优化云中的资源时，作业会持续到达，并且没有明确的终止状态。但是当优化 SQL 连接的顺序时，查询有有限数量的连接，因此经过足够多的步骤，代理到达终止状态。

3.1 Continuing Tasks

An important feature of RL is that it can learn from sparse reward signals, does not need expert labels, and the ability to learn direction from its own interaction with the world. Jobs in the cloud arrive in an unpredictable and continuous manner. This might explain why many system optimization challenges tackled with RL are in the cloud (Mao et al., 2016; He et al., 2017a;b; Tesauro et al., 2006; Xu et al., 2017; Liu et al., 2017; Xu et al., 2012; Rao et al., 2009). A good job scheduler in the cloud should make decisions that are good in the long term. Such a scheduler should sometimes forgoz short term gains in an effort to realize greater long term benefits. For example, it might be better to delay a long running job if a short running job is expected to arrive soon. The scheduler should also adapt to variations in the underlying resource performance and scale in the presence of new or unseen workloads combined with large numbers of resources.

RL 的一个重要特征是它可以从稀疏的奖励信号中学习，不需要专家标签，并且能够从自己与世界的互动中学习方向。云中的作业以不可预测和持续的方式到达。这可能解释了为什么用 RL 解决的许多系统优化挑战都在云中 (Mao 等人，2016; He 等人，2017a; b; Tesauro 等人，2006; 徐等人, 2017; 柳等人，2017; 徐等人，2012; 饒等人，2009)。云中好的作业调度程序应该做出长期良好的决策。这样的调度程序有时应该放弃短期收益，以实现更大的长期收益。例如，如果一个短期运行的作业预计很快就会到达，那么最好推迟一个长期运行的作业。调度程序还应该适应底层资源性能的变化，并在新的或看不见的工作负载与大量资源相结合的情况下进行扩展。

These schedulers have a variety of objectives, including minimizing average performance of jobs and optimizing the resource allocation of virtual machines (Mao et al., 2016; Tesauro et al., 2006; Xu et al., 2012; Rao et al., 2009), optimizing data caching on edge devices and base stations (He et al., 2017a;b), and maximizing energy efficiency (Xu et al., 2017; Liu et al., 2017). The RL algorithms used for addressing each system problem are listed in Table 1 lists the RL algorithms used for addressing each problem.

这些调度程序有多种目标，包括最小化作业的平均性能和优化虚拟机的资源分配(Mao等，2016;Tesauro等，2006;徐等，2012;Rao等，2009)，优化边缘设备和基站上的数据缓存(He等，2017a;b)，最大化能源效率(Xu等，2017;Liu等，2017)。表1列出了用于解决每个系统问题的RL算法。

Interestingly, for cloud challenges most works are driven by Q-learning (or the very similar SARSA). In the absence of a complete environmental model, model-free Q-Learning can be used to generate optimal policies. It is able to make predictions incrementally by bootstrapping the current estimate with previous estimates and provide good sample efficiency (Jin et al., 2018). Q-Learning is also characterized by inherent continuous temporal difference behavior where the policy can be updated immediately after each step (not the end of trajectory); something that might be very useful for online adaptation.

有趣的是，对于云挑战，大多数工作都是由Q学习（或非常类似的SARSA）驱动的。 在缺乏完整的环境模型的情况下，可以使用无模型Q-Learning来生成最优策略。 它能够通过用先前的估计引导当前估计来递增地进行预测，并提供良好的样本效率（Jin等，2018）。 Q-Learning的特征还在于固有的连续时间差异行为，其中策略可以在每个步骤之后立即更新（而不是轨迹的结束）; 对于在线改编可能非常有用的东西。

3.2 Episodic Tasks

Due to the sequential nature of decision making in RL, the order of the actions taken has a major impact on the rewards the RL agent collects. The agent can thus learn these patterns and select more rewarding actions. Previous works took advantage of this behavior in RL to optimize congestion control (Jay et al., 2019; Ruffy et al., 2018), decision trees for packet classification (Liang et al., 2019), sequence to SQL/program translation (Zhong et al., 2017; Guu et al., 2017; Liang et al., 2016), ordering of SQL joins (Krishnan et al., 2018; Ortiz et al., 2018; Marcus et al., 2018; 2019), compiler phase ordering (Huang et al., 2019; Kulkarni et al., 2012) and device placement (Addanki et al., 2019; Paliwal et al., 2019).

由于RL中决策的顺序性，所采取行动的顺序对RL代理收集的奖励有重要影响。因此，代理可以学习这些模式并选择更有价值的行为。以前的工作利用RL中的这种行为来优化拥塞控制(Jay等，2019;Ruffy等，2018)，分组分类决策树(Liang等，2019)，序列到SQL/程序翻译(Zhong等，2017);Guu等，2017;Liang等，2016)，SQL join的排序(Krishnan等，2018;Ortiz等，2018;马库斯等人，2018;、编译器相位排序(Huang等，2019;Kulkarni等，2012)和设备放置(Addanki等，2019;Paliwal等，2019)。

After enough steps in these problems, the agent will always arrive at a clear terminating step. For example, in query join order optimization, the number of joins is finite and known from the query. In congestion control – where the routers need to adapt the sending rates to provide high throughput without comprising fairness – the updates are performed on a fixed number of senders/receivers known in advance. These updates combined define one episode. This may explain why there is a trend towards using PG methods for these types of problems, as they don’t require a continuous temporal difference behavior and can often operate in batches of multiple queries. Nevertheless, in some cases, Q-learning is still used, mainly for sample efficiency as the environment step might take a relatively long time.

在这些问题中经过足够的步骤之后，代理总是会到达一个明确的终止步骤。例如，在查询连接顺序优化中，连接的数量是有限的，并且可以从查询中得知。在拥塞控制中——路由器需要调整发送速率以提供高吞吐量而不影响公平性——更新是在预先知道的固定数量的发送方/接收方上执行的。这些更新组合在一起定义了一个事件。这可能解释了为什么对这类问题有使用PG方法的趋势，因为它们不需要连续的时间差异行为，并且通常可以在多个查询中批量操作。然而，在某些情况下，仍然使用Q-learning，主要是为了提高样本效率，因为环境步骤可能需要较长的时间。

To improve the performance of PG methods, it is possible to take advantage of the way the gradient computation is performed. If the environment is not needed to generate the observation, it is possible to save many environment steps. This is achieved by rolling out the whole episode from interacting only with the policy and performing one environment step at the very end. The sum of rewards will be the same as the reward received from this environment step. For example, in query optimization, since the observations are encoded directly from the actions, and the environment is mainly used to generate the rewards, it will be possible to repeatedly perform an action, form the observation directly from this action, and feed it to the policy network. After the end of the episode, the environment can be triggered to get the final reward, which would be the sum of the intermediate rewards. This can significantly reduce the training time.

为了提高PG方法的性能，可以利用梯度计算的方法。如果不需要环境来生成观察，则可以保存许多环境步骤。实现这一点的方法是，从只与策略交互开始展开整个事件，并在最后执行一个环境步骤。奖励的总和将与此环境步骤所获得的奖励相同。例如，在查询优化中，由于观察值直接由操作编码，并且环境主要用于生成奖励，因此可以重复执行一个操作，直接从该操作生成观察值，并将其提供给策略网络。在本集结束后，可以触发环境得到最终的奖励，也就是中间奖励的总和。这可以大大减少培训时间。

3.3 Discussion: Continuous vs. Episodic

Continuous policies can handle both continuous and episodic tasks, while episodic policies cannot. So, for example, Q-Learning can handle all the tasks mentioned in this work, while PG based methods cannot directly handle it without modification. For example, in (Mao et al., 2016), the authors limited the the scheduler window of jobs to M, allowing the agent in every time step to schedule up to M jobs out of all arrived jobs. The authors also discussed this issue of ”bounded time horizon” and hoped to overcome it by using a value network to replace the time dependent baseline. It is interesting to note that prior work on continuous system optimization tasks using non deep RL approaches (Choi et al., 1996; Littman et al., 2013; Boyan et al., 1994; Peng et al., 2015; Jamshidi et al., 2015; Barrett et al., 2013; Arabnejad et al., 2017; Sadeghi et al., 2017; Farahnakian et al., 2014) used Q-Learning.

连续策略可以同时处理连续和情景任务，而情景策略不能。例如，Q-Learning可以处理这项工作中提到的所有任务，而基于PG的方法不需要修改就不能直接处理。例如，在(Mao et al.， 2016)中，作者将作业的调度程序窗口限制为M，允许代理在每个时间步骤中调度所有到达作业中的至多M个作业。作者还讨论了“有界时间范围”的问题，并希望通过使用一个值网络来代替依赖于时间的基线来克服这一问题。值得注意的是，之前使用非深度RL方法进行连续系统优化任务的工作(Choi et al.， 1996;Littman等，2013;Boyan等，1994;彭等，2015;Jamshidi等，2015;Barrett等，2013;Arabnejad等，2017;Sadeghi等，2017;Farahnakian等，2014)使用Q-Learning。

One solution for handling continuing problems without episode boundaries with PG based methods is to define performance in terms of the average rate of reward per time step (Sutton et al., 2018) (Chapter 13.6). Such approaches can help better fit the continuous problems to episodic RL algorithms.

使用基于PG的方法处理没有情节边界的持续问题的一个解决方案是根据每一步的平均奖励率来定义性能(Sutton et al.， 2018)(第13.6章)。这种方法可以更好地将连续问题与情景RL算法相适应。

5 CONSIDERATIONS FOR EVALUATING DEEP RL IN SYSTEM OPTIMIZATION

In this section, we propose a set of questions that can help system optimization researchers determine whether deep RL could be an effective tool in solving their systems optimization challenges.

在本节中，我们提出了一组问题，可以帮助系统优化研究人员确定deep RL是否可以成为解决他们的系统优化挑战的有效工具。

Can the System Optimization Problem Be Modeled by an MDP?

The problem of RL is the optimal control of an MDP. MDPs are a classical formalization of sequential decision making, where actions influence not just immediate rewards but also future states and rewards. This involves delayed rewards and the trade-off between delayed and immediate reward. In MDPs, the new state and new reward are dependent only on the preceding state and action. Given a perfect model of the environment, an MDP can compute the optimal policy. MDPs are typically a straightforward formulation of the system problem, as an agent learns by continually interacting with the system to achieve a particular goal, and the system responds to these interactions with a new state and reward. The agent’s goal is to maximize expected reward over time.

系统优化问题可以用MDP来建模吗?

RL问题是MDP的最优控制问题。MDPs是一种典型的顺序决策形式化方法，其中的行为不仅影响当前的奖励，还影响未来的状态和奖励。这包括延迟奖励以及延迟奖励和即时奖励之间的权衡。在MDPs中，新的状态和新的奖励只依赖于前面的状态和行为。给定一个完美的环境模型，MDP可以计算出最优策略。MDPs通常是系统问题的一个简单表述，因为代理通过不断地与系统交互来学习以实现特定的目标，并且系统用一个新的状态和奖励来响应这些交互。代理人的目标是随着时间的推移使预期报酬最大化。

Is It a Reinforcement Learning Problem?

What distinguishes RL from other machine learning approaches is the presence of self exploration and exploitation, and the tradeoff between them. For example, RL is different from supervised learning. The latter is learning from a training set with labels provided by an external supervisor that is knowledgeable. For each example the label is the correct action the system should take. The objective of this kind of learning is to act correctly in new situations not present in the training set. However, supervised learning is not suitable for learning from interaction, as it is often impractical to obtain examples representative of all the cases in which the agent has to act.

RL与其他机器学习方法的区别在于自我探索和利用的存在，以及它们之间的权衡。例如，RL不同于监督学习。后者是从由一位知识渊博的外部主管提供的带有标签的培训集中学习的。对于每个示例，标签都是系统应该采取的正确操作。这种学习的目标是采取正确的新情况没有出现在训练集。然而,监督学习不适合学习互动,因为它通常是不切实际的获取例子代表所有的情况下,代理已经采取行动。

9 FUTURE DIRECTIONS

We see multiple future directions for the deployment of deep RL in system optimization tasks. The general assumption is that deep RL may be useful in every system problem where the problem can be formulated as a sequential decision making process, and where meaningful action, state, and reward definitions can be provided. The objective of deep RL in such systems may span a wide range of options, such as energy efficiency, power, reliability, monitoring, revenue, performance, and utilization. At the processor level, deep RL could be used in branch prediction, memory prefetching, caching, data alignment, garbage collection, thread/task scheduling, power management, reliability, and monitoring. Compilers may also benefit from using deep RL to optimize the order of passes (optimizations), knobs/pragmas, unrolling factors, memory expansion, function inlining, vectorizing multiple instructions, tiling and instruction selection. With advancement of in- and near-memory processing, deep RL can be used to determine which portions of a workload should be performed in/near memory and which outside the memory.

我们看到了在系统优化任务中部署 deep RL 的多个未来方向。总的假设是，deep RL 可能在每个系统问题中都有用，在这些问题中，问题可以被表述为顺序决策过程，并且有意义的动作、状态、并且可以提供奖励定义。这种系统中的 deep RL 的目标可能涉及多种选择，如能效、功率、可靠性、监控、收入、性能和利用。在处理器级别，deep RL 可以用于分支预测、内存预取、缓存、数据对齐、垃圾收集、线程/任务调度、电源管理、可靠性和监控。编译器也可能受益于使用 deep RL 优化通过顺序 (优化) 、旋钮/编译器、展开因素、内存扩展、函数内联、向量化多个指令、平铺和指令选择。随着内存内和内存附近处理的推进，deep RL 可以用来确定工作负载的哪些部分应该在内存中/内存附近执行，哪些应该在内存之外执行。

At a higher system level, deep RL may be used in SQL/pandas query optimization, cloud computing, scheduling, caching, monitoring (e.g., temperature/failure) and fault tolerance, packet routing and classification, congestion control, FPGA allocation, and algorithm selection. While some of this has already been done, we believe there is big potential for improvement. It is necessary to explore more benchmarks, stable and generalizable learners, transfer learning approaches, RL algorithms, model-based RL and, more importantly, to provide better encoding of the states, actions and rewards to better represent the system and thus improve the learning. For example, with SQL/pandas join order optimization, the contents of the database are critical for determining the best order, and thus somehow incorporating an encoding of these contents may further improve the performance.

在更高的系统级别，深度RL可用于SQL / pandas查询优化，云计算，调度，缓存，监控（例如，温度/故障）和容错，分组路由和分类，拥塞控制，FPGA分配和算法选择。 虽然其中一些已经完成，但我们认为有很大的改进潜力。 有必要探索更多的基准，稳定和普遍的学习者，转移学习方法，RL算法，基于模型的RL，更重要的是，提供更好的状态，行动和奖励编码，以更好地代表系统，从而改善学习。 例如，使用SQL / pandas连接顺序优化，数据库的内容对于确定最佳顺序是至关重要的，因此以某种方式结合这些内容的编码可以进一步改善性能。

There is room for improvement in the RL algorithms as well. Some action and state spaces can dynamically change with time. For example, when adding a new node to a cluster, the RL agent will always skip the added node and it will not be captured in the environment state. Generally, the state transition function of the environment is unknown to the agent. Therefore, there is no guarantee that if the agent takes a certain action, a certain state will follow in the environment. This issue was presented in (Kulkarni et al., 2012), where compiler optimization passes were selected using deep RL. The authors mentioned a situation where the agent is stuck in an infinite loop of repeatedly picking the same optimization (action) back to back. This issue arose when a particular optimization did not change the features that describe the state of the environment, causing the neural network to apply the same optimization. To break this infinite loop, the authors limited the number of repetitions to five, and then instead, applied the second best optimization. This was done by taking the actions that corresponds to the second highest probability from the neural network’s probability distribution output.

RL算法也有改进的余地。某些动作和状态空间可以随时间动态变化。例如，在向集群添加新节点时，RL代理将始终跳过添加的节点，并且不会在环境状态下捕获该节点。通常，代理不知道环境的状态转换功能。因此，无法保证如果代理采取某种行动，环境中将遵循某种状态。这个问题在（Kulkarni等，2012）中提出，其中使用深RL选择了编译器优化通道。作者提到了一种情况，即代理人陷入无休止的循环中，反复选择相同的优化（动作）背靠背。当特定优化未改变描述环境状态的特征时，会出现此问题，从而导致神经网络应用相同的优化。为了打破这种无限循环，作者将重复次数限制为5次，然后应用第二次最佳优化。这是通过从神经网络的概率分布输出中获取与第二高概率相对应的动作来完成的。

10 CONCLUSION

In this work, we reviewed and discussed multiple challenges in applying deep reinforcement learning to system optimization problems and proposed a set of metrics that can help evaluate the effectiveness of these solutions. Recent applications of deep RL in system optimization are mainly in packet classification, congestion control, compiler optimization, scheduling, query optimization and cloud computing. The growing complexity in systems demands learning based approaches. Deep RL presents unique opportunity to address the dynamic behavior of systems. Applying deep RL to systems proposes new set of challenges on how to frame and evaluate deep RL techniques. We anticipate that solving these challenges will enable system optimization with deep RL to grow.

在这项工作中，我们回顾和讨论了将深度强化学习应用于系统优化问题的多重挑战，并提出了一组指标，可以帮助评估这些解决方案的有效性。最近 deep RL 在系统优化中的应用主要有分组分类、拥塞控制、编译器优化、调度、查询优化和云计算。系统日益复杂，需要基于学习的方法。Deep RL 提供了解决系统动态行为的独特机会。将 deep RL 应用于系统对如何构建和评估 deep RL 技术提出了新的挑战。我们预计，解决这些挑战将使 deep RL 的系统优化得以发展。