Resource Usage Evaluation of Discrete Model-Free Deep Reinforcement Learning Algorithms

Olivia P. Dizon-Paradis*, Stephen E. Wormald, Daniel E. Capecci, Avanti Bhandarkar, and Damon L. Woodard†
Florida Institute for National Security, Electrical and Computer Engineering Department
University of Florida, Gainesville, FL, USA
Email: *paradiso@ufl.edu, †dwoodard@ece.ufl.edu

Abstract

Deep Reinforcement Learning (DRL) has become popular due to promising results in chatbot, healthcare, and autonomous driving applications. However, few DRL algorithms are rigorously evaluated in terms of their space or time efficiency, making them difficult to develop and deploy in practice. In current literature, existing performance comparisons mostly focus on inference accuracy, without considering real-world limitations such as maximum runtime and memory. Furthermore, many works do not make their code publicly accessible for others to use. This paper addresses this gap by presenting the most comprehensive resource usage evaluation and performance comparison of DRL algorithms known to date. This work focuses on publicly-accessible discrete model-free DRL algorithms because of their practicality in real-world problems where efficient implementations are necessary. Although there are other state-of-the art algorithms, few were presently deployment-ready for training on a large number of environments. In total, sixteen DRL algorithms were trained in 23 different environments (468 seeds total), which collectively required 256 GB and 830 CPU days to run all experiments and 1.8 GB to store all models. Overall, our results validate several known challenges in DRL, including exploration and memory inefficiencies, the classic exploration-exploitation trade-off, and large resource utilizations. To address these challenges, this paper suggests numerous opportunities for future work to help improve the capabilities of modern algorithms. The findings of this paper are intended to aid researchers and practitioners in improving and employing DRL algorithms in time-sensitive and resource-constrained applications such as economics, cybersecurity, robotics, and the Internet of Things.

1 Introduction

Lately, Deep Reinforcement Learning (DRL) has become widespread due to the growing popularity of deep neural networks, the rise of big data, and the overwhelming success in various applications (Li, 2017). For example, OpenAI (2023)'s GPT-4 is fine-tuned using DRL from human feedback to answer questions, summarize information, and translate text (Christiano et al., 2017; Uc-Cetina et al., 2023; Liu et al., 2023). In healthcare, DRL has been used to automatically diagnose medical conditions and develop drugs and treatment regimes (Yu et al., 2021). In the autonomous driving domain, DRL has been used to optimize navigation, estimate safety and risk, and predict intentions of pedestrians and other vehicles (Kiran et al., 2021). DRL has also been used in other domains such as finance, advertising, and games (Fischer, 2018; Zhao et al., 2019; Fürnkranz, 2001; Shao et al., 2019). Overall, DRL plays a critical role in many areas in the public and private sectors.

However, few DRL algorithms have been rigorously evaluated, and hence understood, in terms of their time and memory utilizations, which makes it difficult for others to understand which algorithms, if any, can be practically deployed for a given use case. Although there exist some

performance comparisons, many only report inference accuracy (e.g., return) without considering practical limitations such as runtime and memory constraints, and do not publicly release their code. To address these challenges, this paper presents a comprehensive resource usage evaluation and performance comparison of several popular DRL algorithms.

This paper primarily concerns *publicly-available discrete model-free* DRL algorithms. Although there are other state-of-the art algorithms, few are deployment-ready for training on a large number of environments and, hence, would likely not be adopted in a real-world scenario. Discrete DRL was chosen because of its popularity in real-life applications due to the reduced computational complexity of finite actions spaces, versus the infinite action spaces in continuous DRL (Smart & Kaelbling, 2000). Model-free DRL is used instead of model-based DRL because of the former's ease of implementation and tuning in environments without a ground-truth model, which is not available in most practical applications (Sutton & Barto, 2018). Due to these reasons, publicly-inaccessible, continuous, and model-based DRL algorithms are considered out of scope for this study. To our knowledge, this paper presents the most exhaustive resource usage evaluation and performance comparison of DRL algorithms to date, with the following contributions:

- Performance analysis of sixteen DRL algorithms in twenty-three different base environments (468 environment seeds), considering rewards, runtimes, and memory usages
- Open challenges, recommendations for future work, and practical implications
- Re-implemented source code available to facilitate future benchmarking endeavors, collaboration, and development of new technologies[1]

The rest of this paper is organized as follows. Sec. 2 overviews the related works to motivate this study. Sec. 3 describes the performance analysis methodologies. Sec. 4 presents the results and discussion. Sec. 5 presents open challenges, recommendations, and practical implications. Finally, the paper is concluded in Sec. 6 with the key takeaways and future works.

2 Related Works

Prior works related to this study are described in the following subsections. First, existing reviews and comparisons of DRL algorithms are discussed to identify research gaps and highlight the importance of the proposed work. The next subsection introduces `bsuite`, the RL environment suite underpinning this study. Finally, an overview of the existing algorithms used in this paper's methodology is provided.

2.1 Existing Reviews and Comparisons

There are a variety of literature reviews and application surveys in DRL literature that are rich in theory. Literature reviews such as Mousavi et al. (2018), Li (2017), Arulkumaran et al. (2017), Dayan & Niv (2008), and Glorennec (2000) compare DRL algorithms using mathematical equations and pseudocode. Application surveys such as Esteso et al. (2022), Noaeen et al. (2022), den Hengst et al. (2020), and Tran-Dang et al. (2022) investigate the applications of different DRL algorithms across various domains using keyword analyses and bibliometric studies. This paper seeks to complement these theoretical works from a practical perspective by offering a resource usage evaluation and performance comparison with experimental results.

Although there exist some performance comparisons in the DRL literature, most works focus only on inference accuracy (e.g. reward) and rarely report runtimes or memory usages (Fujimoto et al., 2019; Stone et al., 2021; Lin et al., 2021). Of the few works that have addressed time and space efficiencies, Wang et al. (2019) focused on model-based DRL whereas Duan et al. (2016) focused on continuous DRL, which are both considered impractical in many real-world scenarios, for reasons discussed in Sec. 1. Moreover, very few existing works in DRL literature publicly release their source code. This study seeks to build upon the existing works by offering a robust resource usage

[1]source code available at https://github.com/olivia-dizon-paradis/RLPerformanceAnalysis

evaluation and performance comparison of DRL algorithms, and publicly release the source code for others to incorporate into their frameworks.

2.2 Behavior Suite for Reinforcement Learning

This work uses DeepMind's Behaviour Suite for RL (`bsuite`), i.e. the "MNIST of RL", which is a publicly-available suite of twenty-three different base environments (468 seeds total) (Osband et al., 2019a). Although there are a variety of other environment suites, such as Brockman et al. (2016)'s OpenAI Gym classic control environments, Todorov et al. (2012)'s Multi-Joint dynamics with Contact (MuJoCo), Bellemare et al. (2013)'s and Machado et al. (2018)'s Atari 2600, Tassa et al. (2018)'s DeepMind Control Suite, and Cobbe et al. (2020)'s Procgen, `bsuite` was found to be the most comprehensive in testing core RL capabilities for fundamental research and generalization. Among the twenty-three different base `bsuite` environments, 468 total unique environment seeds, or variations, are defined. Each environment is categorized into seven different tags, or categories, based on core challenges in DRL: 'basic', 'noise' , 'scale', 'exploration', 'credit assignment', 'memory', and 'generalization' (Osband et al., 2020). A summary of all environments can be found in Table 1.

Table 1: Overview of the 23 different bsuite environments (Osband et al., 2020).

Environment	Tags[a]	# Actions	# Episodes[b]	Observation Shape[c]	i^d
bandit	bas	11	10000	(1, 1)	20
bandit_noise	noi	11	10000	(1, 1)	20
bandit_scale	sca	11	10000	(1, 1)	20
cartpole	bas, cre, gen	3	1000	(1, 6)	20
cartpole_noise	gen, noi	3	1000	(1, 6)	20
cartpole_scale	gen, sca	3	1000	(1, 6)	20
cartpole_swingup	exp, gen	3	1000	(1, 8)	20
catch	bas, cre	3	10000	(10, 5)	20
catch_noise	cre, noi	3	10000	(10, 5)	20
catch_scale	cre, sca	3	10000	(10, 5)	20
deep_sea	exp	2	10000	$\{(2n, 2n) \mid 5 \leq n \leq 25\}$	21
deep_sea_stochastic	exp,noi	2	10000	$\{(2n, 2n) \mid 5 \leq n \leq 25\}$	21
discounting_chain	cre	5	1000	(1, 2)	20
memory_len	mem	2	10000	(1, 3)	23
memory_size	mem	2	10000	$\{(1, n) \mid 3 \leq n \leq 42, logspaced\}$	17
mnist	bas, gen	10	10000	(28, 28)	20
mnist_noise	gen, noi	10	10000	(28, 28)	20
mnist_scale	gen, sca	10	10000	(28, 28)	20
mountain_car	bas, gen	3	1000	(1, 3)	20
mountain_car_noise	gen, noi	3	1000	(1, 3)	20
mountain_car_scale	gen, sca	3	1000	(1, 3)	20
umbrella_distract	cre, noi	2	10000	$\{(1, n) \mid 4 \leq n \leq 103, logspaced\}$	23
umbrella_length	cre, noi	2	10000	(1, 23)	23
TOTAL					**468**

[a] Environment tags (i.e. categories), shortened to the first three letters
[b] Number of episodes an agent is trained on each seed
[c] Shape of the observation tensor describing the environment's state. Note, for all sets in this column, $n \in N$
[d] Number of unique environment seeds (i.e. variations)

2.3 Existing Deep Reinforcement Learning Algorithms

Given the abundance of different DRL algorithms and variations, this paper focuses on several popular and representative algorithms for each of the main paradigms, especially those with publicly available code. For example, traditional Q-learning methods, i.e. value-based methods that aim to optimize the expected return based on a function of the expected immediate reward, include DQN, Double DQN, and Dueling DQN (Mnih et al., 2013; Van Hasselt et al., 2016; Wang et al., 2016). Distributional Q-learning algorithms, which build on traditional Q-learning models by incorporating full distributions instead of scalar expectations for reward calculations, include Categorical Deep Q-Network (C51) and Rainbow DQN (Bellemare et al., 2017; Hessel et al., 2018). Quantile methods, which further build upon the distributional Q-Learning algorithms by modeling the reward distribu-

tions as quantile functions rather than probability mass functions, include Implicit Quantile Network (IQN) and Fully-parameterized Quantile Function (FQF) (Dabney et al., 2018; Yang et al., 2019). Policy gradient algorithms, which are policy-based methods that explicitly build a mapping between states and actions, include REINFORCE and Natural Policy Gradient (NPG) (Sutton et al., 1999; Kakade, 2001). Actor-critic methods, which combine the value-based and policy-based approaches by using both an "actor" to estimate a policy and a "critic" to estimate the value function, include Synchronous Advantage Actor-Critic (A2C), Proximal Policy Optimization (PPO), and Discrete Soft Actor-Critic (SAC) (Mnih et al., 2016; Schulman et al., 2017; Christodoulou, 2019). In addition, there are also several algorithms specifically designed to address certain challenges in RL, such as Boot DQN (which aids exploration) and A2C RNN (which improves memory). Boot DQN is a DQN variant adapted specifically for exploration problems by using bootstrapping methods to approximate action-value distributions (Osband et al., 2016; 2019b; 2018). A2C RNN is an A2C variant adapted specifically for memory (or exploitation) problems, through the use of a recurrent network to approximate temporal relationships (Williams, 1992; Hochreiter & Schmidhuber, 1997). There are a variety of different ways to classify different DRL algorithms, and many algorithms combine different approaches so they may belong in multiple groups. Here, algorithms are grouped based on their base code implementations for written clarity and ease of reading, particularly for the next section. Although there are certainly other state-of-the art algorithms, very few have publicly accessible code or are not yet deployment-ready for training on a massive number of environments at this time.

3 Methodology

In this study, sixteen DRL algorithms were trained in 23 different base environments (468 seeds), resulting in a total of 7,488 trained agents. Programs were run using Python 3.8.16 (2022), Osband et al. (2020)'s `bsuite` v0.3.5, and Weng et al. (2022)'s Tianshou v0.5.0. Experiments were conducted on NVIDIA GeForce 2080Ti nodes, each with a cyclic allocation of 16GB CPU and 11GB GPU for processing. In total, it took 256 GB and 830 days CPU time (i.e., sixty-nine days on a twelve-node parallel system) to run all experiments and 1.8 GB to store all models.

3.1 Reinforcement Learning Algorithms

Implementation details for the sixteen DRL algorithms used in this study are described in the following subsections. Unless otherwise stated, this work primarily uses algorithm implementations from Weng et al. (2022)'s Tianshou framework. Although there are a variety of other state-of-the-art methods and libraries such as Hill et al. (2018)'s and Raffin et al. (2021)'s Stable Baselines, Kuhnle et al. (2017)'s Tensorforce, Liang et al. (2018)'s RLlib, Pardo (2020b)'s Tonic, Hoffman et al. (2020)'s Acme, Huang et al. (2022)'s CleanRL, and D'Eramo et al. (2021)'s MushroomRL, Tianshou was found to offer the best balance between the number of supported algorithms, training time, and memory consumption at the time of writing given the sheer number of environment seeds and the limited available resources. Although most architectures and hyperparameters were kept faithful to their original base implementations, some parameters among similar algorithms (e.g., discount factor for DQN and double DQN) were set so agent training methodologies were more consistent.

3.1.1 Traditional Q-Learning

Deep Q Network (DQN), Double DQN, and Dueling DQN were implemented with a feedforward multilayer perceptron (MLP) with two sixty-four-unit hidden layers and reLU activation functions. The Dueling DQN's action value (Q) and state value (V) heads each had an additional thirty-two-unit hidden layer. For all Q-learning models, the discount factor was set to 0.99 and the number of steps to look ahead was set to one.

3.1.2 Distributional Q-Learning

Similar to the traditional Q-learning methods, Categorical Deep Q-Network (C51) and Rainbow DQN were both implemented with the same MLP architecture, discount factor, and number of steps to look ahead. For both distributional Q learning methods, the number of atoms (or "canonical returns") was set to the recommended value of fifty-one, with the values of the smallest and largest atoms set to negative and positive ten, respectively.

3.1.3 Quantile

Implicit Quantile Network (IQN) was implemented using a thirty-two-unit hidden layer quantile Q-Network with a double sixty-four-unit hidden layer preprocess MLP. Fully-parameterized Quantile Function (FQF) was implemented using a quantile Q-Network similar to IQN, but with an additional fraction proposal network set to propose thirty-two fractions. For both quantile methods, the discount factor, the number of steps to look ahead, and the number of cosines to use for cosine embedding were set to 0.99, 1, and 64, respectively.

3.1.4 Policy Gradient

REINFORCE Policy Gradient was implemented with a similar MLP and discount factor as the traditional Q-Learning methods. Natural Policy Gradient (NPG) was implemented with thirty-two-unit hidden layer actor and critic modules, where each module consisted of a double sixty-four-unit hidden layer preprocess MLP. For both policy gradient methods, a "categorical" distribution was used for computing the actions.

3.1.5 Actor-Critic

Synchronous Advantage Actor-Critic (A2C), Proximal Policy Optimization (PPO), and Discrete Soft Actor-Critic (SAC) were all implemented with a similar actor-critic architecture to NPG. In addition, A2C's and PPO's discount factor, value loss weight, entropy loss weight, and action distribution were set to 0.99, 0.5, 0.01, and "categorical", respectively. Discrete SAC's discount factor, τ parameter for soft update of the target network, and entropy regularization coefficient were set to 0.99, 0.005, and 0.2, respectively. Moreover, Discrete SAC was designed with an additional critic module, i.e., it had one actor and two critics in total.

3.1.6 bsuite Baselines

In addition, five `bsuite` baselines were re-implemented from Osband et al. (2020) for comparison: four from `baselines/tf` (A2C, DQN, Boot DQN, and A2C RNN) and the randomly-acting agent from `baselines/random`. A2C was implemented with a double sixty-four-unit hidden layer policy value net. DQN was implemented with a double fifty-unit hidden layer MLP. Boot DQN was implemented with a twenty-network ensemble architecture, with each network defined with a similar architecture as the `bsuite` DQN baseline. A2C RNN was implemented with a double sixty-four-unit hidden recurrent layers. As a simple, naive baseline for comparison, experiments were also run using a randomly-acting agent, which randomly picked actions with equal probability. Although bootstrapping and recurrency may be implemented for other algorithms, this paper focused on the algorithms implemented in Osband et al. (2020) at the time of writing for baseline comparison.

Agents were trained using the protocol described in Osband et al. (2019a) and tensorflow baseline scripts in `bsuite` (Osband et al., 2020), which defines the number of training episodes for each environment seed. To ensure the same experiment runner code could be used for DRL algorithms from different libraries, minor changes were made to variable and object names. Agents were trained with an Adam optimizer and random seed of 42 for the random number generator. A summary of the DRL algorithms used in this study can be found in Table 2.

Table 2: Architecture summary of reinforcement learning models used in this study.

Type	Model	# Params[a]	lr[c]	buffer[d]	General Architecture[e]
Traditional Q-Learning	DQN	5k-164k	3e-4	1e4	MLP
	Double DQN	5k-164k	3e-4	1e4	MLP
	Dueling DQN	9k-168k	3e-4	1e4	MLP + (ActionValueHead + StateValueHead)
Distributional Q-Learning	C51	11k-171k	3e-4	1e4	MLP + 51 Atoms
	Rainbow	11k-171k	3e-4	1e4	MLP + 51 Atoms
Quantile	IQN	5k-165k	3e-4	1e4	MLP + CosineEmbedding
	FQF	5k-165k	3e-4	1e4	MLP + CosineEmbedding + FractionProposal
Policy Gradient	REINFORCE	5k-164k	3e-4	1e4	MLP
	NPG	19k-658k	3e-4	1e4	MLP + (Actor + Critic)
Actor-Critic	A2C	19k-658k	3e-4	1e4	MLP + (Actor + Critic)
	PPO	19k-658k	3e-4	1e4	MLP + (Actor + Critic)
	SAC	24k-823k	3e-4	1e4	MLP + (Actor + 2 Critics)
bsuite Baselines	A2C	5k-164k	3e-3	32	MLP + (Actor + Critic)
	DQN	3k-128k	1e-3	1e4	MLP
	Boot DQN	114k-5108k[b]	1e-3	1e4	20 MLPs
	A2C RNN	38k-197k	3e-3	32	RNN + MLP + (Actor + Critic)

[a] Range of trainable parameters, i.e. size of the agent trained on the smallest/largest state-action space environment
[b] Boot DQN is an ensemble of multiple DQNs, with each individual DQN possessing 6k-255k trainable parameters
[c] Learning rate for the Adam optimizer
[d] Buffer size
[e] Although environments varied, the general internal architectures for each algorithm were kept consistent

3.2 Evaluation

To evaluate the resource usage and performance of the DRL agents, this study considers runtime, memory usage, and inference accuracy. Runtime was measured in terms of the wall time needed to train each agent, with the timer starting after agent/environment initialization, and ending before model-saving, testing, and evaluation. Memory usage was measured in terms of the amount of space needed to initialize, update, and save the different agent models. Inference accuracy was computed using `bsuite`'s evaluation framework, by averaging their results across various episodes (Osband et al., 2020). For ease of optimization, performances for most environments were computed as functions of normalized regret scores, i.e. the difference between the payoff of an agent's action and the payoff of the optimal action. For one episode, the `bsuite` performance score, β, was computed for most environments as [2]:

$$100 \times \left(1 - \frac{1}{T}\sum_{t=0}^{T}\frac{r_t^* - r_t}{r^*}\right), \tag{1}$$

where T is the maximum number of timesteps per episode, r_t^* is the reward for taking the best possible action at timestep t, and r_t is the reward the agent actually received at timestep t. Here, obtained reward is subtracted from and divided by the optimal payout, so scores are normalized to allow comparison of performances among environments with different optimal payouts. This term is then divided by T so scores are normalized to allow comparison of performances among environments with different maximum timesteps. Finally, the term is subtracted from one and multiplied by one hundred so results are scaled between zero and one hundred, with the latter defined as the optimal value. To provide insight into each DRL agent's overall performances, results are summarized in the following section by environment tag.

4 Results and Discussion

The inference accuracy, runtime, and memory utilization for each algorithm are summarized by tag in Tables 3, 4a and 4b, respectively. In these tables, "bas", "noi", "sca", "exp", "cre", "mem", and "gen" are three-letter abbreviations for each of the seven tags, as listed in Sec. 2.2.

[2]Note: Although "most environments" were evaluated this way, a different scoring function needed to be used for exploration environments (e.g. `deep sea`, `deep sea stochastic`, and `cartpole swingup`) to prevent penalizing agents for exploring. See Osband et al. (2019a) for more details.

4.1 Inference Accuracy

Quantitatively, the top performing agents in our experiments were as follows. For the 'basic,' 'noise,' 'scale,' and 'generalization' environments, the top performers were `bsuite` DQN and then Boot DQN, with the next-best algorithms trailing by about twenty to thirty points. For the 'exploration' and 'memory' environments, Boot DQN and A2C RNN performed best, respectively, while other agents produced relatively insignificant scores. For the 'credit_assignment' environments, the top performing agent was `bsuite` DQN, with Boot DQN, Double DQN, and `tianshou` DQN trailing by about ten points. Overall, `bsuite` DQN outperformed in all environment categories except 'exploration' and 'memory,' where Boot DQN and A2C RNN performed best, respectively.

There are several potential explanations for the observed quantitative results. The `bsuite` implementations of DQN and Boot DQN, which both encouraged exploration, likely performed well overall because many environments required some level of exploration to prevent from getting stuck in locally optimal, but globally suboptimal, solutions. Boot DQN likely outperformed in the exploration environments because its use of the bootstrap method encouraged higher levels of exploration. For the same reason, Boot DQN likely performed poorly in the memory environments because they did not require (i.e. penalize) heavy exploration. A2C RNN likely performed the best in the memory environments because the recurrent RNN backbone facilitated its ability to approximate and exploit temporal relationships. For the same reason, A2C RNN likely performed poorly in the exploration environments because they did not require (i.e. penalize) heavy exploitation. The traditional Q-learning algorithms also performed well overall, aside from the exploration and memory environments, because the epsilon-greedy algorithm incorporated within their implementations likely encouraged some degree of exploration. Discrete SAC achieved decent results similar to the traditional Q-learning methods and generally outperformed the other actor-critic methods (e.g., A2C and PPO), likely due to Discrete SAC's ability to more accurately estimate the impact of an agent's actions due to the addition of an extra critic module. The distributional Q-learning algorithms and quantile algorithms underperformed, likely because there were not enough features and environment interactions relative to the number of trainable parameters to properly model the value distributions. The policy gradient algorithms generally underperformed, likely due to the relative lack of effective exploration mechanisms in the implementations to prevent agents from getting stuck in locally optimal solutions. The random agent, which was implemented as a naive baseline, performed the worst as expected because it was not designed to learn anything. In general, the `bsuite` variants of DQN and A2C performed better than the `tianshou` ones, likely due to implementation differences in the weight update, action selection, or buffer interaction protocols in the implementation back-ends. Overall, many of the observed results were consistent with theoretical expectations, with some of the more complex models likely underperforming due to the limited number of interactions with the environment relative to the number of trainable parameters. Quantitative results indicate future opportunities for techniques to improve sample efficiency and tune hyperparameters.

Qualitatively, our experiments showed that the most difficult environment categories for the DRL agents overall were 'exploration' and 'memory'. Although Boot DQN performed relatively well in the exploration environments, it performed poorly in the memory ones. The reverse is true for A2C RNN. Of the algorithms tested in this study, none performed well in both exploration and memory environments. Although the results of this study are snapshots of the capabilities of DRL algorithms, they validate several known challenges in applying DRL in real-world scenarios. Many application scenarios are significantly more complex than the toy environments within `bsuite`, as they may involve both memory and exploration elements, require higher inference accuracy with less variation, or limit the number of interactions from which an agent can learn. Overall, inference accuracy results indicate that existing algorithms show promise for simple applications, but that additional research may be required to adapt them for practical ones which are generally more complex.

Algorithm/Tag	bas	noi	sca	exp	cre	mem	gen
DQN	51 ± 25	26 ± 20	31 ± 18	0 ± 0	47 ± 25	2 ± 2	28 ± 19
Double DQN	49 ± 28	25 ± 18	33 ± 20	3 ± 4	48 ± 27	2 ± 2	28 ± 18
Dueling DQN	44 ± 36	22 ± 21	24 ± 19	0 ± 0	40 ± 30	2 ± 2	23 ± 24
C51	4 ± 4	4 ± 4	4 ± 4	3 ± 4	1 ± 2	0 ± 0	5 ± 4
Rainbow	4 ± 4	3 ± 3	7 ± 5	2 ± 2	3 ± 5	0 ± 0	5 ± 3
IQN	28 ± 35	18 ± 18	21 ± 19	2 ± 2	35 ± 31	2 ± 2	8 ± 8
FQF	16 ± 17	10 ± 8	16 ± 12	3 ± 4	17 ± 16	2 ± 2	7 ± 5
REINFORCE	20 ± 28	12 ± 15	19 ± 28	2 ± 2	8 ± 8	2 ± 2	7 ± 3
NPG	8 ± 3	4 ± 4	6 ± 3	0 ± 0	7 ± 8	2 ± 2	6 ± 3
A2C	30 ± 34	15 ± 24	17 ± 16	6 ± 4	12 ± 10	2 ± 2	6 ± 3
PPO	28 ± 35	14 ± 26	17 ± 16	3 ± 4	11 ± 9	2 ± 2	6 ± 3
Discrete SAC	50 ± 32	26 ± 24	17 ± 9	0 ± 0	43 ± 28	2 ± 2	18 ± 15
Random (bsuite)	4 ± 4	2 ± 3	4 ± 4	0 ± 0	3 ± 7	0 ± 0	6 ± 3
A2C (bsuite)	52 ± 37	30 ± 21	28 ± 26	0 ± 0	46 ± 22	2 ± 2	16 ± 15
DQN (bsuite)	77 ± 28	49 ± 31	63 ± 27	0 ± 0	60 ± 25	2 ± 2	53 ± 36
Boot DQN (bsuite)	73 ± 31	44 ± 26	55 ± 25	33 ± 16	49 ± 28	2 ± 2	47 ± 31
A2C RNN (bsuite)	51 ± 38	24 ± 22	23 ± 21	0 ± 0	40 ± 23	50 ± 15	13 ± 14

* The top 33% scores are highlighted in blue, while the bottom 33% are in red.

Table 3: Average inference accuracy (and standard deviation) for each DRL algorithm

Algorithm/Tag	bas	noi	sca	exp	cre	mem	gen
DQN	12	116	14	549	18	19	22
Double DQN	12	113	14	545	18	19	21
Dueling DQN	11	118	15	558	20	23	23
C51	22	90	21	338	26	36	30
Rainbow	19	86	21	338	27	33	29
IQN	21	126	23	563	25	30	38
FQF	40	159	40	633	51	64	64
REINFORCE	16	106	17	487	16	15	24
NPG	103	367	102	1273	150	148	138
A2C	38	185	37	752	55	70	52
PPO	44	199	47	804	58	75	65
Discrete SAC	43	188	45	777	39	39	60
Random (bsuite)	9	9	10	14	4	3	15
A2C (bsuite)	40	37	38	79	21	24	67
DQN (bsuite)	90	107	78	285	98	86	157
Boot DQN (bsuite)	458	958	514	2902	598	620	772
A2C RNN (bsuite)	47	39	40	61	30	21	69

(a) Runtimes (in minutes)

Algorithm/Tag	bas	noi	sca	exp	cre	mem	gen
DQN	64	82	64	199	28	22	81
Double DQN	64	82	64	199	28	22	81
Dueling DQN	82	101	82	217	46	40	99
C51	142	141	142	230	67	48	148
Rainbow	142	141	142	230	67	48	148
IQN	69	87	70	203	32	25	87
FQF	71	89	71	204	33	26	88
REINFORCE	64	82	64	199	28	22	81
NPG	74	93	75	208	37	31	92
A2C	74	93	75	208	37	31	92
PPO	74	93	75	208	37	31	92
Discrete SAC	210	264	210	611	97	79	262
Random (bsuite)	0	0	0	0	0	0	0
A2C (bsuite)	66	84	66	201	29	24	83
DQN (bsuite)	48	63	48	154	20	15	62
Boot DQN (bsuite)	1896	2478	1896	6119	764	583	2447
A2C RNN (bsuite)	198	217	198	333	162	156	216

(b) Memory utilization (in KB)

[a] The fastest 33% runtimes are shown in blue, while the slowest 33% runtimes are in red
[b] The smallest 33% memory usages are in blue, while the largest 33% memory usages are in red

Table 4: Average resource usages for each DRL algorithm

4.2 Runtime and Memory Utilization

In addition to inference accuracy, it is important to discuss the resources utilized for the top-performing algorithms, notably `bsuite` DQN, Boot DQN and A2C RNN. Quantitatively, the training times and memory requirements to store all models were two days and 51 MB for DQN, twenty-three days and 1022 MB for Boot DQN, and one day and 97 MB for A2C RNN. For many applications, these training times are long, as even one day can significantly increase development and deployment times.

Many of the possible reasons the top algorithms performed well in our experiments are likely to be the same reasons that they had such high resource usages. For example, the `bsuite` variants of DQN were slower than the `tianshou` ones despite using less memory, likely due to implementation differences in the weight update, action selection, or buffer interaction protocols in the implementation backends that added additional computations per step. Boot DQN's ensemble structure significantly increased the number of model parameters that needed to be processed and stored, and its use of the bootstrapping algorithm to amalgamate ensemble results increased the agents' training time at every step. For A2C RNN, the increase in space and time complexity is likely due to the additional parameters in the recurrent network and the time required to run recurrent operations. Relative to the other algorithms, the traditional Q-learning agents had lower overall time and space complexities due to their simple MLP structures, but had relatively high time and space complexities in the exploration and noise environments because their model sizes scaled directly with the size of the state-action spaces. Of the actor-critic algorithms, Discrete SAC's extra critic allowed it to generally outperformed A2C and PPO, but also increased its space complexity due to the extra parameters. Consistent with expectation, the randomly acting agent ran fastest and consumed the least amount of space, since it was implemented as a naive baseline that was not designed to learn anything.

Moreover, it is important to consider that a twelve-node system similar to the one we used, with each node allocated 16GB CPU and 11GB GPU for processing, is often not available in practice. Many modern commercial servers may only support four such nodes. On a four-node system with 64GB CPU and 44GB GPU total, DQN, Boot DQN, and A2C RNN may take approximately seven, seventy, and three days, respectively, to train. Training times increase significantly when considering robotics or IoT devices, which may only support one 16GB CPU/11GB GPU node. On such a device, DQN, Boot DQN, and A2C RNN may take around 27, 279, and 14 days to train. For most applications, these training times are unacceptable, as the increased development and deployment times would crucially impact time-to-market, revenue, and deliverable deadlines.

Resource usage results from our experiments validate that most algorithms generally struggled in exploration and memory environments. The runtimes and memory usages for the top-performing algorithms were high. This presents numerous challenges in practical DRL applications, especially those in which exploration or memory are important aspects, computational resources may be limited, and decisions must be made quickly.

5 Open Challenges and Practical Implications

Based on this study's results, one can validate several known challenges in DRL and envision several opportunities for future work. In the following subsections, open challenges are discussed to facilitate collaboration between practitioners and researchers and guide development of new DRL technologies.

5.1 Exploration Inefficiencies

In our experiments, most agents struggled with the exploration environments in general. The one algorithm that showed promising exploration results, Boot DQN, had a very high runtime and memory utilization. This exploration and resource trade-off limits the effectiveness and application of DRL in practical applications where the most rewards are found in the unknown. Future work can focus on characterizing and improving this trade-off, e.g. efficiently adapting rewards to encourage

agents to explore and incorporating traditional path optimization algorithms. If addressed, efficient exploration benefits practical applications where time is critical and decisions must be made quickly, such as deep-sea salvaging and medical drug discovery.

5.2 Memory Inefficiencies

Most agents in this study also struggled with the memory environments; while the one algorithm that showed promising results, A2C RNN, had a high memory utilization and moderately high runtime. This trade-off between memory and resources limits the effectiveness and application of DRL in practical applications where the past significantly affects the future, especially the far future. Future work should focus on characterizing and improving this trade-off, e.g., data compression, importance sampling and weighting, data selection, and attention. If successful, efficient memory benefits applications where the amount of temporal data is particularly massive and resources may be limited, such as economics and cyber-security.

5.3 Exploration-Exploitation Trade-off

It is also important to consider the two previous challenges together. In our experiments, no agent performed well in both exploration and memory environments. This finding validates a classic, yet persistent problem in DRL: the exploration-exploitation trade-off. To help address this problem, future research in this area should focus on better quantifying the trade-off and perhaps looking to other areas of machine learning to develop mitigation strategies (e.g., meta-learning, causal AI, iterative theoretic learning). If successful, this research would benefit applications where both exploration and memory are important, such as chatbots and automated driving.

5.4 Resource Usages

In this study, most well-performing algorithms are estimated to take around 64 GB and seven days to train (based on the `bsuite` DQN resource projections on a four-node system in Sec. 4). Although there are some applications where this resource utilization is acceptable, there are many where it is not. Lengthy time and space complexities greatly decrease the ability of practitioners and researchers to quickly develop and deploy their models. Future works in this field should focus on better quantifying and evaluating the trade-off between performance, time, and space complexities for both training and deployment, and perhaps looking into foundational computer science optimization methodologies to help address it, e.g. algorithm and code optimization, approximate computing, parallel computing, and federated learning paradigms. If resource usages are effectively reduced, such technologies would greatly benefit many resource-constrained fields such as robotics or the Internet of Things (IoT), where a large amount of sensor data must be processed very quickly and the device may go offline, and businesses with quick time-to-markets, where high development and deployment times can result in crushing losses in revenue and missed deliverable deadlines.

5.5 Extensions

There are also several opportunities to improve and extend this line of DRL performance comparison research to other areas not examined in this work. For example, algorithms here were trained similarly across the entire suite of environments for consistency. Future works could focus on improved hyperparameter tuning, such as tuning per environment (e.g. larger buffer sizes for memory environments, higher exploration fractions for exploration environments, etc.). Moreover, although the `bsuite` environment benchmark provides great insight into the core capabilities of reinforcement learning algorithms, it would be beneficial for future work to focus on improving the scale and rigor of more practical real-world challenges as well, such as Dulac-Arnold et al. (2021)'s promising Real-World DRL (RWRL) Challenge Framework which is currently in development. As novel DRL algorithms, frameworks, and model variations are developed, future work should also focus on comprehensively testing and evaluating these new works for their practical implications, such as

in Henderson et al. (2017)'s and Pardo (2020a)'s works, which compare implementation variations. Although this work focused on discrete model-free DRL, many other areas of reinforcement learning would benefit from more comprehensive testing and evaluation, such as multi-agent RL, multi-task RL, and meta-RL. Moreover, although the current work evaluates DRL algorithms across different environment categories, these categories were determined by the creators of `bsuite` and were not rigorously quantified. There is great value in future work that more rigorously quantifies the performances of DRL algorithms, as well as the specific challenges within and difficulty levels of different environments. If successful, such research would help engineers and scientists better identify the reinforcement learning qualities of their specific application scenarios and, hence, select appropriate learning algorithm(s) for their use case.

5.5.1 Explainability

Although understanding and improving the performance, time, and space efficiency of different DRL agents is of great importance, it is often more important to understand why agents behave the way they do. As highlighted in a report from the U.S.A National Security Commission on Artificial Intelligence (AI), the development of ethically-designed, trustworthy AI systems, which are robust, explainable, and fair, is essential for operational integrity and adoption (National Security Commission on Artificial Intelligence, 2019). Explainability helps researchers and practitioners gain human-understandable insights from well-performing models, and improve poorly-performing ones. Future research in DRL should focus on developing more tools to make DRL agents more explainable, ideally through the use of inherently interpretable components such as feature-driven correlation and human-friendly prototypes (Kenny et al., 2023). Such inherently interpretable methods are required to build trustworthy systems for applications such as national security, healthcare, and law, where decisions have a substantial direct impact on human lives.

5.5.2 Imitation Learning and Inverse Reinforcement Learning

Although this paper focuses on scenarios where the reward is known, this is often unrealistic for many practical applications. Future research is necessary to comprehensively test and evaluate algorithms that operate in unknown reward situations, such as imitation learning or inverse reinforcement learning (IRL). Possible avenues for this research include: 1) comparisons between performance, time, and space complexities across different types of environments, 2) imitation or behavior prediction effectiveness across different types of DRL agents, and 3) observation imputation effectiveness across different levels of noise and partial observability. If addressed, such extensions into imitation learning and IRL would benefit applications such as cybersecurity, military, intelligence, reverse-engineering, and inverse goal planning applications where scenarios are constantly changing as each actor seeks an intelligence advantage over the others.

5.6 Inaccessible Code and Data

Perhaps the most important barrier that prevents the adoption of modern DRL techniques is the general lack of shared code and data. As discussed in Sec. 2, many of the state-of-the-art DRL methods remain mainly theoretical. Before anything else, more researchers and practitioners in this field should commit to sharing code and data to mitigate duplication of work, e.g. public release of repositories or supplemental information upon paper acceptance, proper documentation with example scripts and set-up instructions, and publishing all findings including limitations. If more resources were shared, it would facilitate development of new technologies and increase collaboration among researchers and practitioners in industry and academia.

6 Conclusions and Future Work

The key takeaways from this paper are as follows. Deep Reinforcement Learning (DRL), an area of trial-and-error deep learning, has shown promising performances in a variety of difficult applications

within the public and private sectors. However, one of the main challenges in DRL research today is the difficulty in understanding which DRL algorithms are practical for a given use case because many algorithms are not thoroughly tested or evaluated in terms of runtime or memory usage. Therefore, this paper presented the most comprehensive resource evaluation and performance comparison on the practicality of DRL algorithms to date.

Empirical results found the top-performing algorithm overall was `bsuite` DQN for all but the exploration-intensive and memory-intensive environments, where Boot DQN and A2C RNN performed best, respectively. Overall, results indicated that many studied algorithms struggled the most in exploration and memory environments. Moreover, the top performing algorithms had high runtimes or memory utilizations. Such high resource usages are not practical for many real-life applications, as the ensuing increase in development and deployment times can significantly affect time-to-market, revenue, and deliverable deadlines in industry and academia. Although many challenges in practical DRL were validated, from exploration and memory inefficiencies to the classic trade-off between exploration an exploitation, these challenges present numerous opportunities for future work. Efficient resource usage and public availability of code and data can greatly increase the reliability and transparency of DRL research, and hence the operational integrity and adoption of modern DRL algorithms. In future work, the authors seek to extend this work and tackle challenges presented in this paper. If successful, this work can help impact a variety of real-life applications, such as self-driving cars, economics, and cybersecurity.

References

Python 3.8.16. Python Release Python 3.8.16, 12 2022. URL https://www.python.org/downloads/release/python-3816/.

Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47: 253–279, 2013.

Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International conference on machine learning*, pp. 449–458. PMLR, 2017.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

Petros Christodoulou. Soft actor-critic for discrete action settings. *arXiv preprint arXiv:1910.07207*, 2019.

Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pp. 2048–2056. PMLR, 2020.

Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. In *International conference on machine learning*, pp. 1096–1105. PMLR, 2018.

Peter Dayan and Yael Niv. Reinforcement learning: the good, the bad and the ugly. *Current opinion in neurobiology*, 18(2):185–196, 2008.

Floris den Hengst, Eoin Martino Grua, Ali el Hassouni, and Mark Hoogendoorn. Reinforcement learning for personalization: A systematic literature review. *Data Sci.*, 3:107–147, 2020.

Carlo D'Eramo, Davide Tateo, Andrea Bonarini, Marcello Restelli, and Jan Peters. Mushroomrl: Simplifying reinforcement learning research. *Journal of Machine Learning Research*, 22(131):1–5, 2021. URL http://jmlr.org/papers/v22/18-056.html.

Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International conference on machine learning*, pp. 1329–1338. PMLR, 2016.

Gabriel Dulac-Arnold, Nir Levine, Daniel J Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, 110(9):2419–2468, 2021.

Ana Esteso, David Peidro, Josefa Mula, and Manuel Díaz-Madroñero. Reinforcement learning applied to production planning and control. *International Journal of Production Research*, pp. 1–18, 2022.

Thomas G Fischer. Reinforcement learning in financial markets-a survey. Technical report, FAU Discussion Papers in Economics, 2018.

Scott Fujimoto, Edoardo Conti, Mohammad Ghavamzadeh, and Joelle Pineau. Benchmarking batch deep reinforcement learning algorithms. *arXiv preprint arXiv:1910.01708*, 2019.

Johannes Fürnkranz. Machine learning in games: A survey. *Machines that learn to play games*, pp. 11–59, 2001.

Pierre Yves Glorennec. Reinforcement learning: An overview. In *Proceedings European Symposium on Intelligent Techniques (ESIT-00), Aachen, Germany*, pp. 14–15. Citeseer, 2000.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. *CoRR*, abs/1709.06560, 2017. URL http://arxiv.org/abs/1709.06560.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. https://github.com/hill-a/stable-baselines, 2018.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Matthew W. Hoffman, Bobak Shahriari, John Aslanides, Gabriel Barth-Maron, Nikola Momchev, Danila Sinopalnikov, Piotr Stańczyk, Sabela Ramos, Anton Raichuk, Damien Vincent, Léonard Hussenot, Robert Dadashi, Gabriel Dulac-Arnold, Manu Orsini, Alexis Jacq, Johan Ferret, Nino Vieillard, Seyed Kamyar Seyed Ghasemipour, Sertan Girgin, Olivier Pietquin, Feryal Behbahani, Tamara Norman, Abbas Abdolmaleki, Albin Cassirer, Fan Yang, Kate Baumli, Sarah Henderson, Abe Friesen, Ruba Haroun, Alex Novikov, Sergio Gómez Colmenarejo, Serkan Cabi, Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Andrew Cowie, Ziyu Wang, Bilal Piot, and Nando de Freitas. Acme: A research framework for distributed reinforcement learning. *arXiv preprint arXiv:2006.00979*, 2020. URL https://arxiv.org/abs/2006.00979.

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022. URL http://jmlr.org/papers/v23/21-1342.html.

Sham M Kakade. A natural policy gradient. *Advances in neural information processing systems*, 14, 2001.

Eoin M Kenny, Mycal Tucker, and Julie Shah. Towards interpretable deep reinforcement learning with human-friendly prototypes. In *The Eleventh International Conference on Learning Representations*, 2023.

B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(6):4909–4926, 2021.

Alexander Kuhnle, Michael Schaarschmidt, and Kai Fricke. Tensorforce: a tensorflow library for applied reinforcement learning. Web page, 2017. URL https://github.com/tensorforce/tensorforce.

Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. RLlib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2018.

Xingyu Lin, Yufei Wang, Jake Olkin, and David Held. Softgym: Benchmarking deep reinforcement learning for deformable object manipulation. In *Conference on Robot Learning*, pp. 432–448. PMLR, 2021.

Yiheng Liu, Tianle Han, Siyuan Ma, Jiayue Zhang, Yuanyuan Yang, Jiaming Tian, Hao He, Antong Li, Mengshen He, Zhengliang Liu, et al. Summary of chatgpt/gpt-4 research and perspective towards the future of large language models. *arXiv preprint arXiv:2304.01852*, 2023.

Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937. PMLR, 2016.

Seyed Sajad Mousavi, Michael Schukat, and Enda Howley. Deep reinforcement learning: an overview. In *Proceedings of SAI Intelligent Systems Conference (IntelliSys) 2016: Volume 2*, pp. 426–440. Springer, 2018.

National Security Commission on Artificial Intelligence. Interim Report. Technical report, National Security Commission on Artificial Intelligence, November 2019. URL https://www.nscai.gov/wp-content/uploads/2021/01/NSCAI-Interim-Report-for-Congress_201911.pdf.

Mohammad Noaeen, Atharva Naik, Liana Goodman, Jared Crebo, Taimoor Abrar, Zahra Shakeri Hossein Abad, Ana L.C. Bazzan, and Behrouz Far. Reinforcement learning in urban network traffic signal control: A systematic literature review. *Expert Systems with Applications*, 199:116830, 2022. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2022.116830. URL https://www.sciencedirect.com/science/article/pii/S0957417422002858.

OpenAI. Gpt-4 technical report, 2023.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. *Advances in neural information processing systems*, 29, 2016.

Ian Osband, John Aslanides, and Albin Cassirer. Randomized prior functions for deep reinforcement learning. *Advances in Neural Information Processing Systems*, 31, 2018.

Ian Osband, Yotam Doron, Matteo Hessel, John Aslanides, Eren Sezener, Andre Saraiva, Katrina McKinney, Tor Lattimore, Csaba Szepesvari, Satinder Singh, et al. Behaviour suite for reinforcement learning. *arXiv preprint arXiv:1908.03568*, 2019a.

Ian Osband, Benjamin Van Roy, Daniel J Russo, Zheng Wen, et al. Deep exploration via randomized value functions. *J. Mach. Learn. Res.*, 20(124):1–62, 2019b.

Ian Osband, Yotam Doron, Matteo Hessel, John Aslanides, Eren Sezener, Andre Saraiva, Katrina McKinney, Tor Lattimore, Csaba Szepesvári, Satinder Singh, Benjamin Van Roy, Richard Sutton, David Silver, and Hado van Hasselt. Behaviour suite for reinforcement learning. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rygf-kSYwH.

Fabio Pardo. Tonic: A deep reinforcement learning library for fast prototyping and benchmarking. *CoRR*, abs/2011.07537, 2020a. URL https://arxiv.org/abs/2011.07537.

Fabio Pardo. Tonic: A deep reinforcement learning library for fast prototyping and benchmarking. *arXiv preprint arXiv:2011.07537*, 2020b.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Kun Shao, Zhentao Tang, Yuanheng Zhu, Nannan Li, and Dongbin Zhao. A survey of deep reinforcement learning in video games. *arXiv preprint arXiv:1912.10944*, 2019.

William D Smart and Leslie Pack Kaelbling. Practical reinforcement learning in continuous spaces. In *ICML*, pp. 903–910, 2000.

Austin Stone, Oscar Ramirez, Kurt Konolige, and Rico Jonschkowski. The distracting control suite - A challenging benchmark for reinforcement learning from pixels. *CoRR*, abs/2101.02722, 2021. URL https://arxiv.org/abs/2101.02722.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109.

Hoa Tran-Dang, Sanjay Bhardwaj, Tariq Rahim, Arslan Musaddiq, and Dong-Seong Kim. Reinforcement learning based resource management for fog computing environment: Literature review, challenges, and open issues. *Journal of Communications and Networks*, 24(1):83–98, 2022. doi: 10.23919/JCN.2021.000041.

Victor Uc-Cetina, Nicolas Navarro-Guerrero, Anabel Martin-Gonzalez, Cornelius Weber, and Stefan Wermter. Survey on reinforcement learning for language processing. *Artificial Intelligence Review*, 56(2):1543–1575, 2023.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. Benchmarking model-based reinforcement learning. *arXiv preprint arXiv:1907.02057*, 2019.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pp. 1995–2003. PMLR, 2016.

Jiayi Weng, Huayu Chen, Dong Yan, Kaichao You, Alexis Duburcq, Minghao Zhang, Yi Su, Hang Su, and Jun Zhu. Tianshou: A highly modularized deep reinforcement learning library. *Journal of Machine Learning Research*, 23(267):1–6, 2022. URL http://jmlr.org/papers/v23/21-1127.html.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Reinforcement learning*, pp. 5–32, 1992.

Derek Yang, Li Zhao, Zichuan Lin, Tao Qin, Jiang Bian, and Tie-Yan Liu. Fully parameterized quantile function for distributional reinforcement learning. *Advances in neural information processing systems*, 32, 2019.

Chao Yu, Jiming Liu, Shamim Nemati, and Guosheng Yin. Reinforcement learning in healthcare: A survey. *ACM Computing Surveys (CSUR)*, 55(1):1–36, 2021.

Xiangyu Zhao, Long Xia, Jiliang Tang, and Dawei Yin. "deep reinforcement learning for search, recommendation, and online advertising: a survey" by xiangyu zhao, long xia, jiliang tang, and dawei yin with martin vesely as coordinator. *ACM SIGWEB Newsletter*, 2019(Spring):1–15, July 2019. ISSN 1931-1435. doi: 10.1145/3320496.3320500. URL http://dx.doi.org/10.1145/3320496.3320500.