Deep Reinforcement Learning that Matters

Peter Henderson,[1*] **Riashat Islam,**[1,2*] **Philip Bachman,**[2]
Joelle Pineau,[1] **Doina Precup,**[1] **David Meger**[1]

[1] McGill University, Montreal, Canada
[2] Microsoft Maluuba, Montreal, Canada
{peter.henderson,riashat.islam}@mail.mcgill.ca, phbachma@microsoft.com
{jpineau,dprecup}@cs.mcgill.ca, dmeger@cim.mcgill.ca

Abstract

In recent years, significant progress has been made in solving challenging problems across various domains using deep reinforcement learning (RL). Reproducing existing work and accurately judging the improvements offered by novel methods is vital to sustaining this progress. Unfortunately, reproducing results for state-of-the-art deep RL methods is seldom straightforward. In particular, non-determinism in standard benchmark environments, combined with variance intrinsic to the methods, can make reported results tough to interpret. Without significance metrics and tighter standardization of experimental reporting, it is difficult to determine whether improvements over the prior state-of-the-art are meaningful. In this paper, we investigate challenges posed by reproducibility, proper experimental techniques, and reporting procedures. We illustrate the variability in reported metrics and results when comparing against common baselines and suggest guidelines to make future results in deep RL more reproducible. We aim to spur discussion about how to ensure continued progress in the field by minimizing wasted effort stemming from results that are non-reproducible and easily misinterpreted.

Introduction

Reinforcement learning (RL) is the study of how an agent can interact with its environment to learn a policy which maximizes expected cumulative rewards for a task. Recently, RL has experienced dramatic growth in attention and interest due to promising results in areas like: controlling continuous systems in robotics (Lillicrap et al. 2015a), playing Go (Silver et al. 2016), Atari (Mnih et al. 2013), and competitive video games (Vinyals et al. 2017; Silva and Chaimowicz 2017). Figure 1 illustrates growth of the field through the number of publications per year. To maintain rapid progress in RL research, it is important that existing works can be easily reproduced and compared to accurately judge improvements offered by novel methods.

However, reproducing deep RL results is seldom straightforward, and the literature reports a wide range of results for the same baseline algorithms (Islam et al. 2017). Reproducibility can be affected by extrinsic factors (e.g. hyperparameters or codebases) and intrinsic factors (e.g. ef-
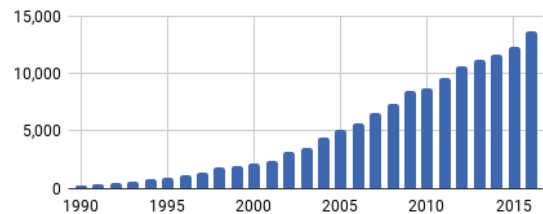
---

*These two authors contributed equally

Figure 1: Growth of published reinforcement learning papers. Shown are the number of RL-related publications (y-axis) per year (x-axis) scraped from Google Scholar searches.

fects of random seeds or environment properties). We investigate these sources of variance in reported results through a representative set of experiments. For clarity, we focus our investigation on policy gradient (PG) methods in continuous control. Policy gradient methods with neural network function approximators have been particularly successful in continuous control (Schulman et al. 2015a; 2017; Lillicrap et al. 2015b) and are competitive with value-based methods in discrete settings. We note that the diversity of metrics and lack of significance testing in the RL literature creates the potential for misleading reporting of results. We demonstrate possible benefits of significance testing using techniques common in machine learning and statistics.

Several works touch upon evaluating RL algorithms. Duan et al. (2016) benchmark several RL algorithms and provide the community with baseline implementations. Generalizable RL evaluation metrics are proposed in (Whiteson et al. 2011). Machado et al. (2017) revisit the Arcade Learning Environment to propose better evaluation methods in these benchmarks. However, while the question of *reproducibility and good experimental practice* has been examined in related fields (Wagstaff 2012; Boulesteix, Lauer, and Eugster 2013; Stodden, Leisch, and Peng 2014; Bouckaert and Frank 2004; Bouckaert 2004; Vaughan and Wawerla 2012), to the best of our knowledge this is the first work to address this important question in the context of deep RL.

In each section of our experimental analysis, we pose questions regarding key factors affecting reproducibility. We find that there are numerous sources of non-determinism when reproducing and comparing RL algorithms. To this end, we

show that fine details of experimental procedure can be critical. Based on our experiments, we conclude with possible recommendations, lines of investigation, and points of discussion for future works to ensure that deep reinforcement learning is reproducible and continues to matter.

Technical Background

This work focuses on several model-free policy gradient algorithms with publicly available implementations which appear frequently in the literature as baselines for comparison against novel methods. We experiment with Trust Region Policy Optimization (TRPO) (Schulman et al. 2015a), Deep Deterministic Policy Gradients (DDPG) (Lillicrap et al. 2015b), Proximal Policy Optimization (PPO) (Schulman et al. 2017), and Actor Critic using Kronecker-Factored Trust Region (ACKTR) (Wu et al. 2017). These methods have shown promising results in continuous control MuJoCo domain tasks (Todorov, Erez, and Tassa 2012) from OpenAI Gym (Brockman et al. 2016). Generally, they optimize $\rho(\theta, s_0) = \mathbb{E}_{\pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t) | s_0 \right]$, using the policy gradient theorem: $\frac{\delta \rho(\theta, s_0)}{\delta \theta} = \sum_s \mu_{\pi_\theta}(s|s_0) \sum_a \frac{\delta \pi_\theta(a|s)}{\delta \theta} Q_{\pi_\theta}(s, a)$. Here, $\mu_{\pi_\theta}(s|s_0) = \sum_{t=0}^{\infty} \gamma^t P(s_t = s|s_0)$ (Sutton et al. 2000). TRPO (Schulman et al. 2015a) and PPO (Schulman et al. 2017) use constraints and advantage estimation to perform this update, reformulating the optimization problem as: $\max_\theta \mathbb{E}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} A_t(s_t, a_t) \right]$. Here, A_t is the generalized advantage function (Schulman et al. 2015b). TRPO uses conjugate gradient descent as the optimization method with a KL constraint: $\mathbb{E}_t \left[\text{KL} \left[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t) \right] \right] \leq \delta$. PPO reformulates the constraint as a penalty (or clipping objective). DDPG and ACKTR use actor-critic methods which estimate $Q(s, a)$ and optimize a policy that maximizes the Q-function based on Monte-Carlo rollouts. DDPG does this using deterministic policies, while ACKTR uses Kronecketer-factored trust regions to ensure stability with stochastic policies.

Experimental Analysis

We pose several questions about the factors affecting reproducibility of state-of-the-art RL methods. We perform a set of experiments designed to provide insight into the questions posed. In particular, we investigate the effects of: specific hyperparameters on algorithm performance if not properly tuned; random seeds and the number of averaged experiment trials; specific environment characteristics; differences in algorithm performance due to stochastic environments; differences due to codebases with most other factors held constant. For most of our experiments[1], except for those comparing codebases, we generally use the OpenAI Baselines[2] implementations of the following algorithms: ACKTR (Wu et al. 2017), PPO (Schulman et al. 2017), DDPG (Plappert et al. 2017), TRPO (Schulman et al. 2017). We use the Hopper-v1 and HalfCheetah-v1 MuJoCo (Todorov, Erez, and Tassa 2012) environments from OpenAI Gym (Brockman et al.

2016). These two environments provide contrasting dynamics (the former being more unstable).

To ensure fairness we run five experiment trials for each evaluation, each with a different preset random seed (all experiments use the same set of random seeds). In all cases, we highlight important results here, with full descriptions of experimental setups and additional learning curves included in the supplemental material. Unless otherwise mentioned, we use default settings whenever possible, while modifying only the hyperparameters of interest.

We use multilayer perceptron function approximators in all cases. We denote the hidden layer sizes and activations as $(N, M, \text{activation})$. For default settings, we vary the hyperparameters under investigation one at a time. For DDPG we use a network structure of $(64, 64, \text{ReLU})$ for both actor and critic. For TRPO and PPO, we use $(64, 64, \text{tanh})$ for the policy. For ACKTR, we use $(64, 64, \text{tanh})$ for the actor and $(64, 64, \text{ELU})$ for the critic.

Hyperparameters

What is the magnitude of the effect hyperparameter settings can have on baseline performance?

Tuned hyperparameters play a large role in eliciting the best results from many algorithms. However, the choice of optimal hyperparameter configuration is often not consistent in related literature, and the range of values considered is often not reported[3]. Furthermore, poor hyperparameter selection can be detrimental to a fair comparison against baseline algorithms. Here, we investigate several aspects of hyperparameter selection on performance.

Network Architecture

How does the choice of network architecture for the policy and value function approximation affect performance?

In (Islam et al. 2017), it is shown that policy network architecture can significantly impact results in both TRPO and DDPG. Furthermore, certain activation functions such as Rectified Linear Unit (ReLU) have been shown to cause worsened learning performance due to the "dying relu" problem (Xu et al. 2015). As such, we examine network architecture and activation functions for both policy and value function approximators. In the literature, similar lines of investigation have shown the differences in performance when comparing linear approximators, RBFs, and neural networks (Rajeswaran et al. 2017). Tables 1 and 2 summarize the final evaluation performance of all architectural variations after training on 2M samples (i.e. 2M timesteps in the environment). All learning curves and details on setup can be found in the supplemental material. We vary hyperparameters one at a time, while using a default setting for all others. We investigate three multilayer perceptron (MLP) architectures commonly seen in the literature: $(64, 64)$, $(100, 50, 25)$, and $(400, 300)$. Furthermore, we vary the activation functions of both the value and policy networks across tanh, ReLU, and Leaky ReLU activations.

Results Figure 2 shows how significantly performance can be affected by simple changes to the policy or value network

---

[1]Specific details can be found in the supplemental and code can be found at: https://git.io/vFHnf

[2]https://www.github.com/openai/baselines

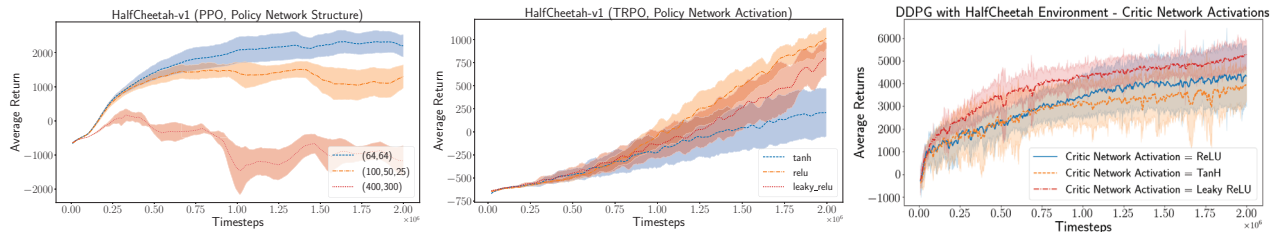[3]A sampled literature review can be found in the supplemental.

Figure 2: Significance of Policy Network Structure and Activation Functions PPO (left), TRPO (middle) and DDPG (right).
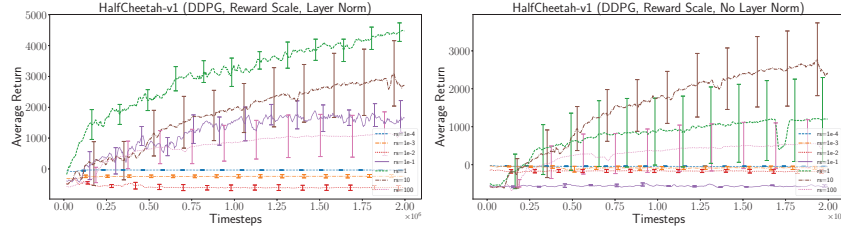

Figure 3: DDPG reward rescaling on HalfCheetah-v1, with and without layer norm.

activations. We find that usually ReLU or Leaky ReLU activations perform the best across environments and algorithms. The effects are not consistent across algorithms or environments. This inconsistency demonstrates how interconnected network architecture is to algorithm methodology. For example, using a large network with PPO may require tweaking other hyperparameters such as the trust region clipping or learning rate to compensate for the architectural change[4]. This intricate interplay of hyperparameters is one of the reasons reproducing current policy gradient methods is so difficult. It is exceedingly important to choose an appropriate architecture for proper baseline results. This also suggests a possible need for hyperparameter agnostic algorithms—that is algorithms that incorporate hyperparameter adaptation as part of the design—such that fair comparisons can be made without concern about improper settings for the task at hand.

Reward Scale

How can the reward scale affect results? Why is reward rescaling used?

Reward rescaling has been used in several recent works (Duan et al. 2016; Gu et al. 2016) to improve results for DDPG. This involves simply multiplying the rewards generated from an environment by some scalar ($\hat{r} = r\hat{\sigma}$) for training. Often, these works report using a reward scale of $\hat{\sigma} = 0.1$. In Atari domains, this is akin to clipping the rewards to $[0, 1]$. By intuition, in gradient based methods (as used in most deep RL) a large and sparse output scale can result in problems regarding saturation and inefficiency in learning (LeCun et al. 2012; Glorot and Bengio 2010; Vincent, de Brébisson, and Bouthillier 2015). Therefore clipping or rescaling rewards compresses the space of estimated

---

[4]We find that the KL divergence of updates with the large network $(400, 300)$ seen in Figure 2 is on average 33.52 times higher than the KL divergence of updates with the $(64, 64)$ network.

expected returns in action value function based methods such as DDPG. We run a set of experiments using reward rescaling in DDPG (with and without layer normalization) for insights into how this aspect affects performance.

Results Our analysis shows that reward rescaling can have a large effect (full experiment results can be found in the supplemental material), but results were inconsistent across environments and scaling values. Figure 3 shows one such example where reward rescaling affects results, causing a failure to learn in small settings below $\hat{\sigma} = 0.01$. In particular, layer normalization changes how the rescaling factor affects results, suggesting that these impacts are due to the use of deep networks and gradient-based methods. With the value function approximator tracking a moving target distribution, this can potentially affect learning in unstable environments where a deep Q-value function approximator is used. Furthermore, some environments may have untuned reward scales (e.g. the HumanoidStandup-v1 of OpenAI gym which can reach rewards in the scale of millions). Therefore, we suggest that this hyperparameter has the potential to have a large impact if considered properly. Rather than rescaling rewards in some environments, a more principled approach should be taken to address this. An initial foray into this problem is made in (van Hasselt et al. 2016), where the authors adaptively rescale reward targets with normalized stochastic gradient, but further research is needed.

Random Seeds and Trials

Can random seeds drastically alter performance? Can one distort results by averaging an improper number of trials?

A major concern with deep RL is the variance in results due to environment stochasticity or stochasticity in the learning process (e.g. random weight initialization). As such, even averaging several learning results together across totally different random seeds can lead to the reporting of misleading results. We highlight this in the form of an experiment.

Algorithm	Environment	400,300	64,64	100,50,25	tanh	ReLU	LeakyReLU
TRPO	Hopper-v1	2980 ± 35	2674 ± 227	3110 ± 78	2674 ± 227	2772 ± 211	-
(Schulman et al. 2015a)	HalfCheetah-v1	1791 ± 224	1939 ± 140	2151 ± 27	1939 ± 140	3041 ± 161	-
TRPO	Hopper-v1	1243 ± 55	1303 ± 89	1243 ± 55	1303 ± 89	1131 ± 65	1341 ± 127
(Duan et al. 2016)	HalfCheetah-v1	738 ± 240	834 ± 317	850 ± 378	834 ± 317	784 ± 352	1139 ± 364
TRPO	Hopper-v1	2909 ± 87	2828 ± 70	2812 ± 88	2828 ± 70	2941 ± 91	2865 ± 189
(Schulman et al. 2017)	HalfCheetah-v1	-155 ± 188	205 ± 256	306 ± 261	205 ± 256	1045 ± 114	778 ± 177
PPO	Hopper-v1	61 ± 33	2790 ± 62	2592 ± 196	2790 ± 62	2695 ± 86	2587 ± 53
(Schulman et al. 2017)	HalfCheetah-v1	-1180 ± 444	2201 ± 323	1314 ± 340	2201 ± 323	2971 ± 364	2895 ± 365
DDPG	Hopper-v1	1419 ± 313	1632 ± 459	2142 ± 436	1491 ± 205	1632 ± 459	1384 ± 285
(Plappert et al. 2017)	HalfCheetah-v1	5579 ± 354	4198 ± 606	5600 ± 601	5325 ± 281	4198 ± 606	4094 ± 233
DDPG	Hopper-v1	600 ± 126	593 ± 155	501 ± 129	436 ± 48	593 ± 155	319 ± 127
(Gu et al. 2016)	HalfCheetah-v1	2845 ± 589	2771 ± 535	1638 ± 624	1638 ± 624	2771 ± 535	1405 ± 511
DDPG	Hopper-v1	506 ± 208	749 ± 271	629 ± 138	354 ± 91	749 ± 271	-
(Duan et al. 2016)	HalfCheetah-v1	850 ± 41	1573 ± 385	1224 ± 553	1311 ± 271	1573 ± 385	-
ACKTR	Hopper-v1	2577 ± 529	1608 ± 66	2287 ± 946	1608 ± 66	2835 ± 503	2718 ± 434
(Wu et al. 2017)	HalfCheetah-v1	2653 ± 408	2691 ± 231	2498 ± 112	2621 ± 381	2160 ± 151	2691 ± 231

Table 1: Results for our policy architecture permutations across various implementations and algorithms. Final average $\pm$ standard error across 5 trials of returns across the last 100 trajectories after 2M training samples. For ACKTR, we use **ELU** activations instead of leaky ReLU.

Algorithm	Environment	400,300	64,64	100,50,25	tanh	ReLU	LeakyReLU
TRPO	Hopper-v1	3011 ± 171	2674 ± 227	2782 ± 120	2674 ± 227	3104 ± 84	-
(Schulman et al. 2015a)	HalfCheetah-v1	2355 ± 48	1939 ± 140	1673 ± 148	1939 ± 140	2281 ± 91	-
TRPO	Hopper-v1	2909 ± 87	2828 ± 70	2812 ± 88	2828 ± 70	2829 ± 76	3047 ± 68
(Schulman et al. 2017)	HalfCheetah-v1	178 ± 242	205 ± 256	172 ± 257	205 ± 256	235 ± 260	325 ± 208
PPO	Hopper-v1	2704 ± 37	2790 ± 62	2969 ± 111	2790 ± 62	2687 ± 144	2748 ± 77
(Schulman et al. 2017)	HalfCheetah-v1	1523 ± 297	2201 ± 323	1807 ± 309	2201 ± 323	1288 ± 12	1227 ± 462
DDPG	Hopper-v1	1419 ± 312	1632 ± 458	1569 ± 453	971 ± 137	852 ± 143	843 ± 160
(Plappert et al. 2017)	HalfCheetah-v1	5600 ± 601	4197 ± 606	4713 ± 374	3908 ± 293	4197 ± 606	5324 ± 280
DDPG	Hopper-v1	523 ± 248	343 ± 34	345 ± 44	436 ± 48	343 ± 34	-
(Gu et al. 2016)	HalfCheetah-v1	1373 ± 678	1717 ± 508	1868 ± 620	1128 ± 511	1717 ± 508	-
DDPG	Hopper-v1	1208 ± 423	394 ± 144	380 ± 65	354 ± 91	394 ± 144	-
(Duan et al. 2016)	HalfCheetah-v1	789 ± 91	1095 ± 139	988 ± 52	1311 ± 271	1095 ± 139	-
ACKTR	Hopper-v1	152 ± 47	1930 ± 185	1589 ± 225	691 ± 55	500 ± 379	1930 ± 185
(Wu et al. 2017)	HalfCheetah-v1	518 ± 632	3018 ± 386	2554 ± 219	2547 ± 172	3362 ± 682	3018 ± 38

Table 2: Results for our value function (Q or V) architecture permutations across various implementations and algorithms. Final average $\pm$ standard error across 5 trials of returns across the last 100 trajectories after 2M training samples. For ACKTR, we use **ELU** activations instead of leaky ReLU.
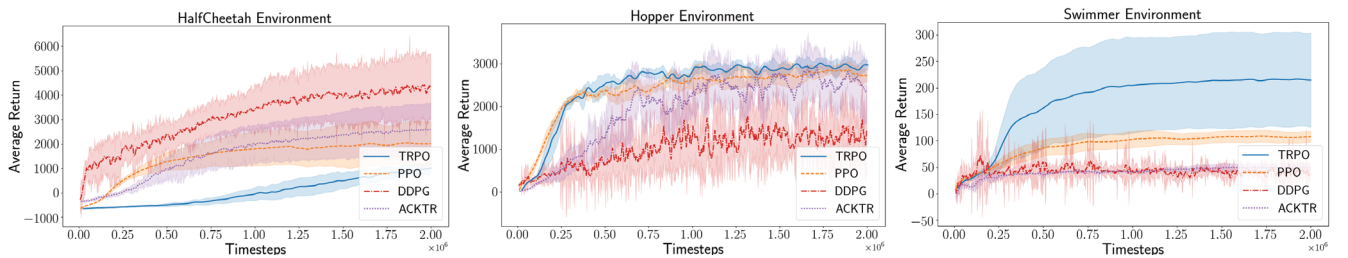

Figure 4: Performance of several policy gradient algorithms across benchmark MuJoCo environment suites

Environment	DDPG	ACKTR	TRPO	PPO
HalfCheetah-v1	5037 (3664, 6574)	3888 (2288, 5131)	1254.5 (999, 1464)	3043 (1920, 4165)
Hopper-v1	1632 (607, 2370)	2546 (1875, 3217)	2965 (2854, 3076)	2715 (2589, 2847)
Walker2d-v1	1582 (901, 2174)	2285 (1246, 3235)	3072 (2957, 3183)	2926 (2514, 3361)
Swimmer-v1	31 (21, 46)	50 (42, 55)	214 (141, 287)	107 (101, 118)

Table 3: Bootstrap mean and 95% confidence bounds for a subset of environment experiments. 10k bootstrap iterations and the pivotal method were used.
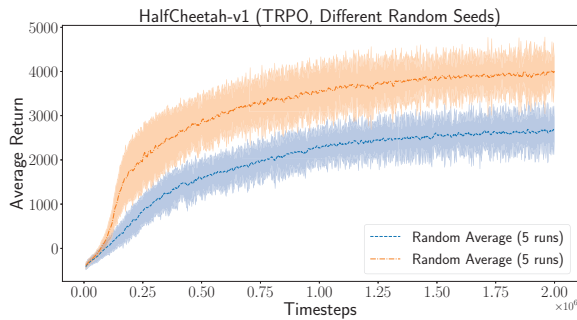
HalfCheetah-v1 (TRPO, Different Random Seeds)

Figure 5: TRPO on HalfCheetah-v1 using the same hyperparameter configurations averaged over two sets of 5 different random seeds each. The average 2-sample t-test across entire training distribution resulted in $t = -9.0916$, $p = 0.0016$.

Results We perform 10 experiment trials, for the same hyperparameter configuration, only varying the random seed across all 10 trials. We then split the trials into two sets of 5 and average these two groupings together. As shown in Figure 5, we find that the performance of algorithms can be drastically different. We demonstrate that the variance between runs is enough to create statistically different distributions just from varying random seeds. Unfortunately, in recent reported results, it is not uncommon for the top-N trials to be selected from among several trials (Wu et al. 2017; Mnih et al. 2016) or averaged over only small number of trials ($N < 5$) (Gu et al. 2017; Wu et al. 2017). Our experiment with random seeds shows that this can be potentially misleading. Particularly for HalfCheetah, it is possible to get learning curves that do not fall within the same distribution at all, just by averaging different runs with the same hyperparameters, but different random seeds. While there can be no specific number of trials specified as a recommendation, it is possible that power analysis methods can be used to give a general idea to this extent as we will discuss later. However, more investigation is needed to answer this open problem.

Environments

How do the environment properties affect variability in reported RL algorithm performance?

To assess how the choice of evaluation environment can affect the presented results, we use our aforementioned default set of hyperparameters across our chosen testbed of algorithms and investigate how well each algorithm performs across an extended suite of continuous control tasks. For these experiments, we use the following environments from OpenAI Gym: Hopper-v1, HalfCheetah-v1, Swimmer-v1 and Walker2d-v1. The choice of environment often plays an important role in demonstrating how well a new proposed algorithm performs against baselines. In continuous control tasks, often the environments have random stochasticity, shortened trajectories, or different dynamic properties. We demonstrate that, as a result of these differences, algorithm performance can vary across environments and the best performing algorithm across all environments is not always clear. Thus it is

increasingly important to present results for a wide range of environments and not only pick those which show a novel work outperforming other methods.

Results As shown in Figure 4, in environments with stable dynamics (e.g. HalfCheetah-v1), DDPG outperforms all other algorithsm. However, as dynamics become more unstable (e.g. in Hopper-v1) performance gains rapidly diminish. As DDPG is an off-policy method, exploration noise can cause sudden failures in unstable environments. Therefore, learning a proper Q-value estimation of expected returns is difficult, particularly since many exploratory paths will result in failure. Since failures in such tasks are characterized by shortened trajectories, a local optimum in this case would be simply to survive until the maximum length of the trajectory (corresponding to one thousand timesteps and similar reward due to a survival bonus in the case of Hopper-v1). As can be seen in Figure 4, DDPG with Hopper does exactly this. This is a clear example where showing only the favourable and stable HalfCheetah when reporting DDPG-based experiments would be unfair.

Furthermore, let us consider the Swimmer-v1 environment shown in Figure 4. Here, TRPO significantly outperforms all other algorithms. Due to the dynamics of the water-like environment, a local optimum for the system is to curl up and flail without proper swimming. However, this corresponds to a return of $\sim$130. By reaching a local optimum, learning curves can indicate successful optimization of the policy over time, when in reality the returns achieved are not qualitatively representative of learning the desired behaviour, as demonstrated in video replays of the learned policy[5]. Therefore, it is important to show not only returns but demonstrations of the learned policy in action. Without understanding what the evaluation returns indicate, it is possible that misleading results can be reported which in reality only optimize local optima rather than reaching the desired behaviour.

Codebases

Are commonly used baseline implementations comparable?

In many cases, authors implement their own versions of baseline algorithms to compare against. We investigate the OpenAI baselines implementation of TRPO as used in (Schulman et al. 2017), the original TRPO code (Schulman et al. 2015a), and the rllab (Duan et al. 2016) Tensorflow implementation of TRPO. We also compare the rllab Theano (Duan et al. 2016), rllabplusplus (Gu et al. 2016), and OpenAI baselines (Plappert et al. 2017) implementations of DDPG. Our goal is to draw attention to the variance due to implementation details across algorithms. We run a subset of our architecture experiments as with the OpenAI baselines implementations using the same hyperparameters as in those experiments[6].

Results We find that implementation differences which are often not reflected in publications can have dramatic impacts on performance. This can be seen for our final evaluation performance after training on 2M samples in Tables 1

---

[5]https://youtu.be/lKpUQYjgm80

[6]Differences are discussed in the supplemental (e.g. use of different optimizers for the value function baseline). Leaky ReLU activations are left out to narrow the experiment scope.
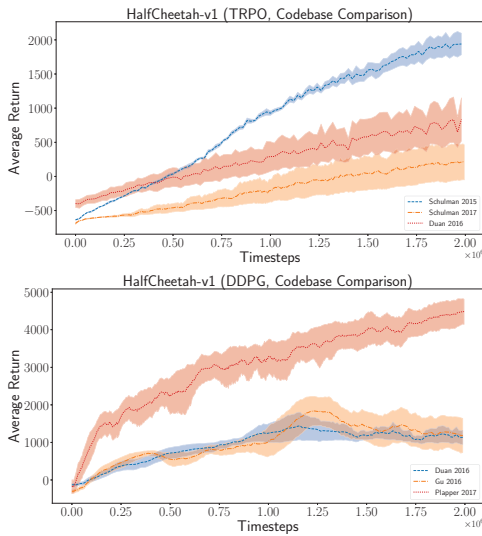
Figure 6: TRPO codebase comparison using our default set of hyperparameters (as used in other experiments).

and 2, as well as a sample comparison in Figure 6. This demonstrates the necessity that implementation details be enumerated, codebases packaged with publications, and that performance of baseline experiments in novel works matches the original baseline publication code.

Reporting Evaluation Metrics

In this section we analyze some of the evaluation metrics commonly used in the reinforcement learning literature. In practice, RL algorithms are often evaluated by simply presenting plots or tables of average cumulative reward (average returns) and, more recently, of maximum reward achieved over a fixed number of timesteps. Due to the unstable nature of many of these algorithms, simply reporting the maximum returns is typically inadequate for fair comparison; even reporting average returns can be misleading as the range of performance across seeds and trials is unknown. Alone, these may not provide a clear picture of an algorithm's range of performance. However, when combined with confidence intervals, this may be adequate to make an informed decision given a large enough number of trials. As such, we investigate using the bootstrap and significance testing as in ML (Kohavi and others 1995; Bouckaert and Frank 2004; Nadeau and Bengio 2000) to evaluate algorithm performance.

Online View vs. Policy Optimization An important distinction when reporting results is the online learning view versus the policy optimization view of RL. In the online view, an agent will optimize the returns across the entire learning process and there is not necessarily an end to the agent's trajectory. In this view, evaluations can use the average cumulative rewards across the entire learning process (balancing exploration and exploitation) as in (Hofer and Gimbert 2016), or can possibly use offline evaluation as in (Mandel et al. 2016). The alternate view corresponds to policy optimization, where evaluation is performed using a target policy in an of-

fline manner. In the policy optimization view it is important to run evaluations across the entire length of the task trajectory with a single target policy to determine the average returns that the target can obtain. We focus on evaluation methods for the policy optimization view (with offline evaluation), but the same principles can be applied to the online view.

Confidence Bounds The sample bootstrap has been a popular method to gain insight into a population distribution from a smaller sample (Efron and Tibshirani 1994). Bootstrap methods are particularly popular for A/B testing, and we can borrow some ideas from this field. Generally a bootstrap estimator is obtained by resampling with replacement many times to generate a statistically relevant mean and confidence bound. Using this technique, we can gain insight into what is the 95% confidence interval of the results from our section on environments. Table 3 shows the bootstrap mean and 95% confidence bounds on our environment experiments. Confidence intervals can vary wildly between algorithms and environments. We find that TRPO and PPO are the most stable with small confidence bounds from the bootstrap. In cases where confidence bounds are exceedingly large, it may be necessary to run more trials (i.e. increase the sample size).

Power Analysis Another method to determine if the sample size must be increased is bootstrap power analysis (Tufféry 2011; Yuan and Hayashi 2003). If we use our sample and give it some uniform lift (for example, scaling uniformly by 1.25), we can run many bootstrap simulations and determine what percentage of the simulations result in statistically significant values with the lift. If there is a small percentage of significant values, a larger sample size is needed (more trials must be run). We do this across all environment experiment trial runs and indeed find that, in more unstable settings, the bootstrap power percentage leans towards insignificant results in the lift experiment. Conversely, in stable trials (e.g. TRPO on Hopper-v1) with a small sample size, the lift experiment shows that no more trials are needed to generate significant comparisons. These results are provided in the supplemental material.

Significance An important factor when deciding on an RL algorithm to use is the significance of the reported gains based on a given metric. Several works have investigated the use of significance metrics to assess the reliability of reported evaluation metrics in ML. However, few works in reinforcement learning assess the significance of reported metrics. Based on our experimental results which indicate that algorithm performance can vary wildly based simply on perturbations of random seeds, it is clear that some metric is necessary for assessing the significance of algorithm performance gains and the confidence of reported metrics. While more research and investigation is needed to determine the best metrics for assessing RL algorithms, we investigate an initial set of metrics based on results from ML.

In supervised learning, k-fold t-test, corrected resampled t-test, and other significance metrics have been discussed when comparing machine learning results (Bouckaert and Frank 2004; Nadeau and Bengio 2000). However, the assumptions pertaining to the underlying data with corrected metrics do not necessarily apply in RL. Further work is needed to investigate proper corrected significance tests for RL. Nonetheless,

we explore several significance measures which give insight into whether a novel algorithm is truly performing as the state-of-the-art. We consider the simple 2-sample t-test (sorting all final evaluation returns across N random trials with different random seeds); the Kolmogorov-Smirnov test (Wilcox 2005); and bootstrap percent differences with 95% confidence intervals. All calculated metrics can be found in the supplemental. Generally, we find that the significance values match up to what is to be expected. Take, for example, comparing Walker2d-v1 performance of ACKTR vs. DDPG. ACKTR performs slightly better, but this performance is not significant due to the overlapping confidence intervals of the two: $t = 1.03, p = 0.334, KS = 0.40, p = 0.697$, bootstrapped percent difference 44.47% (-80.62%, 111.72%).

Discussion and Conclusion

Through experimental methods focusing on PG methods for continuous control, we investigate problems with reproducibility in deep RL. We find that both intrinsic (e.g. random seeds, environment properties) and extrinsic sources (e.g. hyperparameters, codebases) of non-determinism can contribute to difficulties in reproducing baseline algorithms. Moreover, we find that highly varied results due to intrinsic sources bolster the need for using proper significance analysis. We propose several such methods and show their value on a subset of our experiments.

What recommendations can we draw from our experiments?

Based on our experimental results and investigations, we can provide some general recommendations. Hyperparameters can have significantly different effects across algorithms and environments. Thus it is important to find the working set which at least matches the original reported performance of baseline algorithms through standard hyperparameter searches. Similarly, new baseline algorithm implementations used for comparison should match the original codebase results if available. Overall, due to the high variance across trials and random seeds of reinforcement learning algorithms, many trials must be run with different random seeds when comparing performance. Unless random seed selection is explicitly part of the algorithm, averaging multiple runs over different random seeds gives insight into the population distribution of the algorithm performance on an environment. Similarly, due to these effects, it is important to perform proper significance testing to determine if the higher average returns are in fact representative of better performance.

We highlight several forms of significance testing and find that they give generally expected results when taking confidence intervals into consideration. Furthermore, we demonstrate that bootstrapping and power analysis are possible ways to gain insight into the number of trial runs necessary to make an informed decision about the significance of algorithm performance gains. In general, however, the most important step to reproducibility is to report *all* hyperparameters, implementation details, experimental setup, and evaluation methods for both baseline comparison methods and novel work. Without the publication of implementations and related details, wasted effort on reproducing state-of-the-art works will plague the community and slow down progress.

What are possible future lines of investigation?

Due to the significant effects of hyperparameters (particularly reward scaling), another possibly important line of future investigation is in building hyperparameter agnostic algorithms. Such an approach would ensure that there is no unfairness introduced from external sources when comparing algorithms agnostic to parameters such as reward scale, batch size, or network structure. Furthermore, while we investigate an initial set of significance metrics here, they may not be the best fit for comparing RL algorithms. Several works have begun investigating policy evaluation methods for the purposes of safe RL (Thomas and Brunskill 2016; Thomas, Theocharous, and Ghavamzadeh 2015), but further work is needed in significance testing and statistical analysis. Similar lines of investigation to (Nadeau and Bengio 2000; Bouckaert and Frank 2004) would be helpful to determine the best methods for evaluating performance gain significance.

How can we ensure that deep RL matters?

We discuss many different factors affecting reproducibility of RL algorithms. The sensitivity of these algorithms to changes in reward scale, environment dynamics, and random seeds can be considerable and varies between algorithms and settings. Since benchmark environments are proxies for real-world applications to gauge generalized algorithm performance, perhaps more emphasis should be placed on the applicability of RL algorithms to real-world tasks. That is, as there is often no clear winner among all benchmark environments, perhaps recommended areas of application should be demonstrated along with benchmark environment results when presenting a new algorithm. Maybe new methods should be answering the question: in what setting would this work be useful? This is something that is addressed for machine learning in (Wagstaff 2012) and may warrant more discussion for RL. As a community, we must not only ensure reproducible results with fair comparisons, but we must also consider what are the best ways to demonstrate that RL continues to matter.

Acknowledgements

References

Bouckaert, R. R., and Frank, E. 2004. Evaluating the replicability of significance tests for comparing learning algorithms. In *PAKDD*, 3–12. Springer.

Bouckaert, R. R. 2004. Estimating replicability of classifier learning experiments. In *Proceedings of the 21st International Conference on Machine Learning (ICML)*.

Boulesteix, A.-L.; Lauer, S.; and Eugster, M. J. 2013. A plea for neutral comparison studies in computational sciences. *PloS one* 8(4):e61562.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. OpenAI gym. *arXiv preprint arXiv:1606.01540*.

Duan, Y.; Chen, X.; Houthooft, R.; Schulman, J.; and Abbeel, P. 2016. Benchmarking deep reinforcement learning for continuous

control. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*.

Efron, B., and Tibshirani, R. J. 1994. *An introduction to the bootstrap*. CRC press.

Glorot, X., and Bengio, Y. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 249–256.

Gu, S.; Lillicrap, T.; Ghahramani, Z.; Turner, R. E.; and Levine, S. 2016. Q-prop: Sample-efficient policy gradient with an off-policy critic. *arXiv preprint arXiv:1611.02247*.

Gu, S.; Lillicrap, T.; Ghahramani, Z.; Turner, R. E.; Schölkopf, B.; and Levine, S. 2017. Interpolated policy gradient: Merging on-policy and off-policy gradient estimation for deep reinforcement learning. *arXiv preprint arXiv:1706.00387*.

Hofer, L., and Gimbert, H. 2016. Online reinforcement learning for real-time exploration in continuous state and action markov decision processes. *arXiv preprint arXiv:1612.03780*.

Islam, R.; Henderson, P.; Gomrokchi, M.; and Precup, D. 2017. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *ICML Reproducibility in Machine Learning Workshop*.

Kohavi, R., et al. 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *IJCAI*, volume 14.

LeCun, Y. A.; Bottou, L.; Orr, G. B.; and Müller, K.-R. 2012. Efficient backprop. In *Neural Networks: Tricks of the Trade*. Springer.

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

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

Machado, M. C.; Bellemare, M. G.; Talvitie, E.; Veness, J.; Hausknecht, M.; and Bowling, M. 2017. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *arXiv preprint arXiv:1709.06009*.

Mandel, T.; Liu, Y.-E.; Brunskill, E.; and Popovic, Z. 2016. Offline Evaluation of Online Reinforcement Learning Algorithms. In *AAAI*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 1928–1937.

Nadeau, C., and Bengio, Y. 2000. Inference for the generalization error. In *Advances in neural information processing systems*.

Plappert, M.; Houthooft, R.; Dhariwal, P.; Sidor, S.; Chen, R.; Chen, X.; Asfour, T.; Abbeel, P.; and Andrychowicz, M. 2017. Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*.

Rajeswaran, A.; Lowrey, K.; Todorov, E.; and Kakade, S. 2017. Towards generalization and simplicity in continuous control. *arXiv preprint arXiv:1703.02660*.

Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; and Moritz, P. 2015a. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*.

Schulman, J.; Moritz, P.; Levine, S.; Jordan, M.; and Abbeel, P. 2015b. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Silva, V. d. N., and Chaimowicz, L. 2017. Moba: a new arena for game ai. *arXiv preprint arXiv:1705.10443*.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Stodden, V.; Leisch, F.; and Peng, R. D. 2014. *Implementing reproducible research*. CRC Press.

Sutton, R. S.; McAllester, D. A.; Singh, S. P.; and Mansour, Y. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*.

Thomas, P., and Brunskill, E. 2016. Data-efficient off-policy policy evaluation for reinforcement learning. In *International Conference on Machine Learning*, 2139–2148.

Thomas, P. S.; Theocharous, G.; and Ghavamzadeh, M. 2015. High-Confidence Off-Policy Evaluation. In *AAAI*.

Todorov, E.; Erez, T.; and Tassa, Y. 2012. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012, Vilamoura, Algarve, Portugal, October 7-12, 2012*, 5026–5033.

Tufféry, S. 2011. *Data mining and statistics for decision making*, volume 2. Wiley Chichester.

van Hasselt, H. P.; Guez, A.; Hessel, M.; Mnih, V.; and Silver, D. 2016. Learning values across many orders of magnitude. In *Advances in Neural Information Processing Systems*, 4287–4295.

Vaughan, R., and Wawerla, J. 2012. Publishing identifiable experiment code and configuration is important, good and easy. *arXiv preprint arXiv:1204.2235*.

Vincent, P.; de Brébisson, A.; and Bouthillier, X. 2015. Efficient exact gradient update for training deep networks with very large sparse targets. In *Advances in Neural Information Processing Systems*, 1108–1116.

Vinyals, O.; Ewalds, T.; Bartunov, S.; Georgiev, P.; Vezhnevets, A. S.; Yeo, M.; Makhzani, A.; Küttler, H.; Agapiou, J.; Schrittwieser, J.; et al. 2017. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*.

Wagstaff, K. 2012. Machine learning that matters. *arXiv preprint arXiv:1206.4656*.

Whiteson, S.; Tanner, B.; Taylor, M. E.; and Stone, P. 2011. Protecting against evaluation overfitting in empirical reinforcement learning. In *2011 IEEE Symposium on Adaptive Dynamic Programming And Reinforcement Learning, ADPRL 2011, Paris, France, April 12-14, 2011*, 120–127.

Wilcox, R. 2005. Kolmogorov–smirnov test. *Encyclopedia of biostatistics*.

Wu, Y.; Mansimov, E.; Liao, S.; Grosse, R.; and Ba, J. 2017. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. *arXiv preprint:1708.05144*.

Xu, B.; Wang, N.; Chen, T.; and Li, M. 2015. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*.

Yuan, K.-H., and Hayashi, K. 2003. Bootstrap approach to inference and power analysis based on three test statistics for covariance structure models. *British Journal of Mathematical and Statistical Psychology* 56(1):93–110.