RE-EVALUATE: Reproducibility in Evaluating Reinforcement Learning Algorithms

Khimya Khetarpal[*], **Zafarali Ahmed**[*], **Andre Cianflone**, **Riashat Islam**, **Joelle Pineau**

Reasoning and Learning Lab
Montreal Institute for Learning Algorithms (Mila)
McGill University, Montreal, Canada

Abstract

Reinforcement learning (RL) has recently achieved tremendous success in solving complex tasks. Careful considerations are made towards reproducible research in machine learning. Reproducibility in RL often becomes more difficult, due to the lack of standard evaluation method and detailed methodology for algorithms and comparisons with existing work. In this work, we highlight key differences in evaluation in RL compared to supervised learning, and discuss specific issues that are often non-intuitive for newcomers. We study the importance of reproducibility in evaluation in RL, and propose an evaluation pipeline that can be decoupled from the algorithm code. We hope such an evaluation pipeline can be standardized, as a step towards robust and reproducible research in RL.

1 Introduction

In recent years, reinforcement learning (RL) combined with neural network approximators has achieved extraordinary success in solving complex tasks: the game of Go [1]; continuous control tasks such as locomotion skills [2]; and planning chemical syntheses [3]. The advent of advanced computing has enabled rapid progress in not only Deep Learning (DL), but also in Deep RL. To ensure consistent progress in the field, reproducibility in research is a vital tool. Recently, significant steps have been made towards ensuring reproducible research in machine learning[2]. However, with Deep RL, reproducible research is often more difficult [4] due to several factors including intrinsic variance of the algorithm, stochasticity of the environment [5] and dependence on several hyperparameters [6]. Ensuring that RL results are robust and reproducible is vital for future research. One crucial aspect of reproducible research is the ability to quickly and fairly evaluate and compare algorithms. Recently, through the efforts of authors and reproducibility challenges, open source implementations of algorithms are becoming more readily available. However, this is only one part of the puzzle: in this work we draw the distinction between *reproducibility in algorithm* vs. *reproducibility in evaluation*, and are mainly concerned with the latter.

For a supervised learning task, the evaluation of an algorithm is standard: the dataset is split into training, validation and testing subsets. The specific held-out portion for the test set is often pre-portioned by the dataset authors, as with MNIST [7], or specified by the paper authors, such as the section breakdown for the Penn Treebank [8]. Such schemes are often made into "challenges". For instance, the MIT Saliency Benchmark [9] and the Imagenet challenge [10] hold out the labeled test set for submissions to be evaluated in a uniform and consistent way. In all aforementioned cases, evaluating a supervised learning algorithm generally consists of simply plotting the negative

[*]Equal contribution

[2]ICLR 2018 Reproducibility Challenge, `http://goo.gl/Xev1qV`

2nd Reproducibility in Machine Learning Workshop at ICML 2018, Stockholm, Sweden.

log-likelihood training curve and reporting accuracy on the test set. In contrast, for RL, this is not so obvious. A recent investigation on overfitting in RL has encouraged the use of different training and testing seeds [11]: RL agents can overfit quite robustly to training instances of maze games motivating an evaluation pipeline. We share such a call and provide one instance of such a pipeline.

The Arcade Learning Environment (ALE) [12] was proposed as a framework to assess general competency of RL agents. While assessment of algorithms on several tasks is key to Artificial General Intelligence (AGI), our foremost emphasis here is: Given a *specific* task, how can we compare algorithms uniformly and consistently? Many published papers in RL provide a wide variety of results on almost the entire suite of ALE games. However, without access to the exact evaluation pipeline, comparing algorithms requires reproducing the evaluation pipeline alongside the implementation itself. In RL, algorithms are often evaluated by *learning curves* performance, but not all algorithms mention the key hyper-parameters leading to the highlights in their results. Hyperparameters in Deep RL can however significantly affect results [4, 6]. RL algorithms are often evaluated with different number of roll-outs for average returns, random seeds, episodes for each run and varying repetitions of each experiments, making comparisons across published results difficult.

While there are many RL tool-kits which provide a suite of environments [12, 13, 14], it is not always straightforward to evaluate several RL algorithms on a same *specific* task. In RL, due to various intrinsic and external factors, comparisons to baseline algorithms from existing work is often an onerous task. OpenAI Baselines [15] is an effort to implement high quality versions of RL algorithms for the community to re-use and allow comparisons. Moreover, the choice of metric for the evaluation is subject to the subtle details of the proposed algorithm and the task at hand. This makes it difficult to reproduce results exactly [4], or to match the evaluation protocol across algorithms [16]. The closest to standard evaluation pipelines are competitions such as Pommerman[3] and Learning to Run[4]. Previously, a standard evaluation scheme existed for OpenAI Gym in the form of a leader board[5] where algorithms could be evaluated using a public pipeline. We speculate the project was no longer maintained due to the high number of submissions and the difficulty in comparing quality research.

In this work, we highlight the challenges for newcomers to the field, in terms of reproducing and comparisons to existing results. We propose the need for a pipeline that can provide consistent evaluations and provide a proof of concept for how we could potentially develop tools for enhancing reproducible RL research. While we do not address the issue of algorithms tested across several tasks, we believe our evaluation pipeline can be extended to transfer learning. Our **key contributions**:

- We illustrate a case-study to highlight the importance of reproducibility in evaluation, by emphasizing the challenges in comparing across different algorithms.
- We define an evaluation pipeline and demonstrate via a prototype how RL algorithms can be evaluated fairly and consistently. We propose that authors should either re-use or release such an evaluation pipeline which will ensure uniform comparison of RL algorithms.
- We show that given a further layer of abstraction, algorithms can be comparable even in the absence of open source code.

2 Illustrative Study

In existing work, evaluating an algorithm usually consists of measuring average return achieved in a given number of time steps, or the sample efficiency (especially when evaluating off-policy algorithms). However, evaluation of RL algorithms greatly vary across different works. For instance, some authors report performance during the learning phase itself after every n rollouts, while others report average score over k once training is stopped, whereas some report scores on variations of the environment not seen during training [17]. In this paper, we argue that such differences in protocol often makes comparison across algorithms difficult. We emphasize that, while sample efficiency as an evaluation metric may be required, few standard metrics should always be used for a fair comparison against baselines. A detailed discussion on how to evaluate algorithms in an environment is important, for example Bellemare et al. [12] provide a thorough analysis of evaluation methodology for the ALE suite of games, along with insights on ways to analyze performance of an algorithm. Here we attempt to present an environment agnostic discussion.

[3] `https://www.pommerman.com/`

[4] `https://www.crowdai.org/challenges/nips-2017-learning-to-run`

[5] `https://github.com/openai/gym/wiki/Leaderboard`

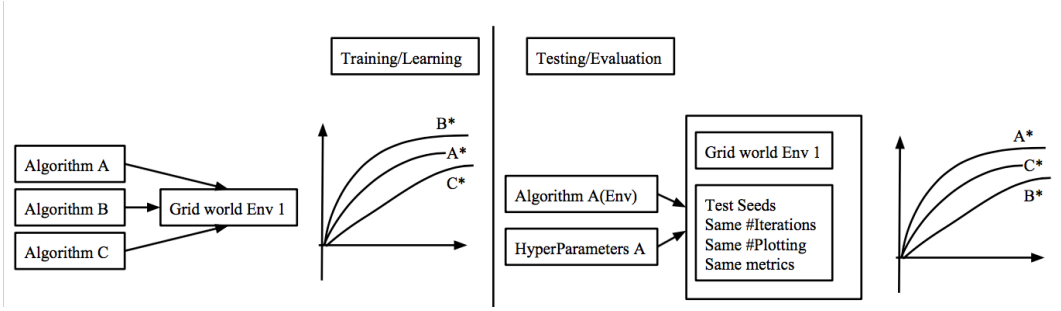

Figure 1: **Overview of the evaluation pipeline:** During the training phase, algorithms are compared against each with out-of-the box settings and weights of the learned model. At test time, the evaluation pipeline provides the ability to feed in a common choice of metric for evaluation and accessibility to the existing algorithms in a reproducible abstraction.

To investigate the issues in evaluation, let us look at a minimal illustrative study here. In the cart-pole domain, the task is considered solved once a cumulative reward of 200 is achieved. Assume we propose a new variant of REINFORCE [18]. We want to compare our proposed algorithm with other open-source baselines, such as the Deep-Q network (DQN) [19] from Keras-RL [20]. We start by setting up the codebases, and running them out-of-the-box. The foremost challenge is each algorithm saves the results in different formats. For instance, at the first glance, by default DQN-Keras-RL saves the weights of the trained model and outputs command line logs comprised of episode rewards, mean rewards and timesteps. On the other hand, our REINFORCE algorithm saves the return for episodes into a json format. A logging standard is required to facilitate comparison across these two results and enable fair evaluation. In both implementations, return per episode is logged and each episode terminates after a fixed number of timesteps. However, there are additional differences such as action-repetition (number of times the agent repeats the same action without observing the environment again), which used in DQN-Keras-RL and may impact results. Ths tool-kit implements a standalone test function independent of the evaluations made during the training phase. This results in metrics such as the average return logged in a different way across algorithms.

If we extended this analysis to also include an algorithm from OpenAI Baselines [15], the above mentioned inconsistencies are even more stark: results are saved into a csv format; further investigation will be made to check for evaluation consistency. This may require us to adapt our code to fit both libraries. After examining many open-source implementations, we found that collecting the required results from multiple algorithms into a common file format may require not only extensive code rewrite, but also significant efforts to comb through the codebases to find out how exactly the evaluation metrics are collected. For a fair comparison of algorithms with baselines, such as by plotting learning curves, we need to develop a tool which takes the output of the specific algorithms and evaluates them. Furthermore, it is not clear what specific hyperparameters were used in the different algorithms, which may make the evaluation and comparison even more difficult.

3 Evaluation Pipeline

In section 2 we illustrated subtle variations in different approaches to evaluating RL algorithms, highlighting the need for an explicit evaluation phase which follows a standard protocol. The key problem lies in *how* and *when* performance is reported. Generally, performance is evaluated by plotting average return over a number of episodes. However, the way in which these returns are collected for plotting purposes throughout the learning varies from one work to another. Moreover, there is no standard way in which the average return is collected for evaluating the algorithm. In other words, different open-source implementations use different standards for collecting the average return during the policy evaluation phase as seen in section 2. Furthermore, often the average return achieved during training phase is shown as a metric of performance.

In this work, we introduce an evaluation pipeline as shown in figure 1. The key to our proposed method is an evaluation mechanism with clear distinction between *train* and *test phases*. Our proposed evaluation pipeline works as follows: consider algorithms A, B and C that need to be compared against each other. These algorithms would be handed off to our evaluation mechanism, along with the environment to test it on. Our proposed mechanism would evaluate these algorithms over a large number of known random seeds for the environment generator and would record standardized metrics. The pipeline would then output the learning performance of each of these algorithms to ensure a fair

comparison where all other metrics remain the same. All that is required is that the algorithm exposes an `act(state)` method.

Input : Environment, E; Algorithm, A; Evaluation Scheme ,M.
Output : Consistent evaluations
$\pi \longleftarrow$ Initialize Policy; // π must expose an `act` method.
while *True* **do**
  A.trainstep(π, E)
  M.evaluate(π); // M uses `act` to record metrics in a uniform way.
end

Algorithm 1: Separating Training and Evaluation in Reinforcement Learning

We propose a simple prototype of our evaluation pipeline as shown in Algorithm 1, and hope that such an approach can be standardized in RL to ensure reproducibility and fair comparison across algorithms. We present our evaluation pipeline as follows: When releasing open-source implementations of algorithms, we suggest each algorithm to be wrapped in a format acceptable to the pipeline. Our pipeline takes 3 key objects as input. The first is the Environment E. This could be a certain Gym [13] or other commonly used environment over which our pipeline can iterate. E would also include seeds, or seed strategy (such as purely random), which the pipeline would report for ease of reproducibility. The second object is the algorithm A being evaluated, such as our REINFORCE implementation. A also includes a configuration file of hyperparameters. Finally, the evaluation metric M specifying how to analyze the algorithm is provided. We propose that the training and evaluation metrics should be done explicitly separately using the $trainstep$ and $evaluate$ modules as shown in the algorithm. We emphasize here that the $trainstep$ is the universal module which is provided by algorithm A as input. The evaluation scheme is provided by the pipeline which would evaluate algorithm A with a given number of rollouts and log all relevant metrics to further analyze algorithm performance. Similarly, this pipeline can also take other baseline algorithms B and C as inputs, and produce consistent evaluations across all algorithms A, B, and C for a fair comparison. We provide a framework agnostic pseudo code depicting this scenario in the appendix. An advantage of the pipeline is that M can be easily swapped for other evaluation schemes. The proposed pipeline along with the implementations of our case study in section 2 is available on our repository.[6]

4 Discussion

In this work, we highlight the challenges in evaluating RL algorithms. We argue that a broad spectrum of evaluation metrics ranging from sample efficiency to average return over last 100 episodes are often used to highlight the usefulness of a certain algorithm, which may make comparisons of it with other baselines more difficult. Evaluation protocols used for RL algorithms vary across published works, especially when there is no distinction between training and test phases in RL. We emphasize that this raises few challenges for newcomers in RL, when implementing and comparing their algorithms with other state of the art methods. To ensure a fair comparison to baseline methods, the use of standardized evaluation metrics, are often key steps towards reproducible research in RL.

In this work, we propose that the community follow a standard evaluation pipeline decoupling the algorithm from the evaluation. We demonstrate *one* way of doing this could be with set arguments (Environment E, Algorithm A, and Evaluation Scheme M) in terms of standard abstractions such as trainstep(), learn(), act() as depicted in the framework agnostic evaluation pipeline. We emphasize that authors should release careful details about evaluation schemes used in their work, which can often lead to new contributions [21, 22]. For instance, time steps at which return is logged, exact specifics of evaluation, detailed description of hyper parameters such as specific seeds, the n in n-step rollouts, would help in reproducing results. As demonstrated by our proof-of-concept, we encourage that each open sourced algorithm should either reuse or release an evaluation pipeline. This allows authors to reuse configurations and results file, while not necessarily having access to algorithm code. It would then become feasible to evaluate a new algorithm and reproduce exiting baselines, thereby bringing consistency in comparisons across algorithms

[6] `https://github.com/kkhetarpal/prototype4evaluation`

References

[1] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[2] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015.

[3] Marwin HS Segler, Mike Preuss, and Mark P Waller. Planning chemical syntheses with deep neural networks and symbolic ai. *Nature*, 555(7698):604, 2018.

[4] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560*, 2017.

[5] Shimon Whiteson, Brian Tanner, Matthew E. Taylor, and Peter Stone. Protecting against evaluation overfitting in empirical reinforcement learning. In *ADPRL 2011: Proceedings of the IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pages 120–127, April 2011.

[6] Riashat Islam, Peter Henderson, Maziar Gomrokchi, and Doina Precup. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *CoRR*, abs/1708.04133, 2017.

[7] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[8] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

[9] Zoya Bylinskii, Tilke Judd, Ali Borji, Laurent Itti, Frédo Durand, Aude Oliva, and Antonio Torralba. Mit saliency benchmark, 2015.

[10] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[11] Chiyuan Zhang, Oriol Vinyals, Remi Munos, and Samy Bengio. A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893*, 2018.

[12] 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.

[13] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[14] Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

[15] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Openai baselines. `https://github.com/openai/baselines`, 2017.

[16] Adrien Lucas Ecoffet. Paper repro: Deep neuroevolution, 2018.

[17] J. Oh, V. Chockalingam, S. Singh, and H. Lee. Control of Memory, Active Perception, and Action in Minecraft. *ArXiv e-prints*, May 2016.

[18] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*, pages 5–32. Springer, 1992.

[19] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[20] Matthias Plappert. keras-rl. `https://github.com/keras-rl/keras-rl`, 2016.

[21] G. Tucker, S. Bhupatiraju, S. Gu, R. E. Turner, Z. Ghahramani, and S. Levine. The Mirage of Action-Dependent Baselines in Reinforcement Learning. *ArXiv e-prints*, February 2018.

[22] Martin Riedmiller, Jan Peters, and Stefan Schaal. Evaluation of policy gradient methods and variants on the cart-pole benchmark. In *Approximate Dynamic Programming and Reinforcement Learning, 2007. ADPRL 2007. IEEE International Symposium on*, pages 254–261. IEEE, 2007.

Appendix

Framework agnostic example scripts

Example 1

Alice implements 3A (Alice Amazing Algorithm) and tests her hyperparameters on grid worlds of size (1, 10)

```
import tensorflow as tf
# some tensorflow code
class AliceAmazingAlgorithm():
        def __init__(self, hyperparameters):
                """
                The 3A algorithm, the best RL algorithm ever (for gridworlds)
                :param hyperparameters: the hyperparameters for the algorithm
                """
                self.state = tf.placeholder()
                Linear_layer = tf.linear()
                self.policy_result = Linear_layer(state)

        def learn(self, environment, iterations):
                Action = sess.run(results, feed_dict={state}
                environment.step(action)
                # policy learning here

        def Act(self, state):
                Return sess.run(results, feed_dict={state}
```

Example 2

Bob implements 2BP (Bob Builds Policies) and tests his hyperparameters on grid worlds of size (10, 20)

```
import torch
# some pytorch code
class BobBuildsPolicies():
        def __init__(self, hyperparameters):
                """
                The 2BO algorithm, the best RL algorithm ever (for gridworlds)
                :param hyperparameters: the hyperparameters for the algorithm
                """
                Self.policy = torch.nn.Linear(bla, bla)

        def learn(self, environment, iterations):
                Loss = do_rollout(environment, self.policy)
                optimizer.zero_grad()
                los.backward()
                optimizer.step()

        def Act(self, state):
                Return self.policy(state)
```

Evaluation Pipeline

How would one benchmark these two algorithms on a set of grid worlds? Both Alice and Bob claim that their algorithms are the best grid world solvers for a specific task in a specific grid world. Without having access to each others evaluation and source code scripts, how would they compare their algorithms one-to-one? In addition to this, it is likely that each use their own crazy ways to plot data. What if there was an agreed upon a standard evaluation script that they could use made available by the makers of a grid world environment?

```
Grid world evaluation pipeline

1. User edits the configuration file
# file name: config.json
{
Algo_path: "path/to/your/algorithm/file'' # user algorithm file path
Algorithm: "main(train_ddpg, 'Algorithm') # bind user algo with benchmark algo
Hyperparameters = load_txt(''path/to/hyperparameters'')
}

2. User launches the benchmarking script
# minimal example of the benchmark script (user cannot edit this)
For env_seed in LIST_OF_AGREED_UPON_SEEDS_FOR_ENV:
        Env = env(env_seed, env_characteristics) # make a new environment
Alg = Algorithm(Hyperparameters)
Alg.learn(Env, 10000)
Reward_curve, other_metrics = do_test_rollout(Alg.act, Env)
np.save([reward_curve, other_metrics], 'evaluation_on_env{n}.npy'.format(env_seed))
```

Now the authors only have to make available the evaluation on env npy files to allow one to one comparison. Therefore, Bob, with access to this script and the evaluated curves can recreate plots to compare with Alice!