Reproducing Softmax Deep Double Deterministic Policy Gradients

Anonymous Author(s)
Affiliation
Address
`email`

Reproducibility Summary

Scope of Reproducibility

We attempt to reproduce Pan et al. [19] claim that Softmax Deep Double Deterministic Policy Gradient (SD3) achieves superior performance over Twin Delayed Deep Double Deterministic Policy Gradient (TD3) [9] on continuous control reinforcement learning tasks. We utilize both environments that were used by the paper and expand to include some not present.

Methodology

We compare the performance of TD3 and SD3 on a variety of continuous control tasks. We use the author's PyTorch code but also provide Tensorflow implementations of SD3 and TD3 (which we did not use for optimization reasons). For the control tasks we utilize OpenAI Gym environments with PyBullet implementations, as opposed to MuJoCo, in an effort to bolster claims of generalization and to avoid exclusionary research practices. Experiments are conducted both on similar environments in the original paper and those that were not mentioned.

Results

Overall we reach similar, albeit much milder, conclusions as the paper, specifically, that SD3 outperforms TD3 on some of continuous control tasks. However, the advantage is not always as readily apparent as in the original work. Algorithmic performance was comparable on most environments, with SD3 providing limited evidence of definitive superiority. Further investigation and improvements are warranted. The results are not directly comparable to the original paper due to differences in physics simulators. Additionally, we did not perform hyperparameter optimization, which could potentially bolster returns on some environments.

What was easy

The author's made their code extremely easy to use, run, modify and rewrite in a different package. Because everything was available on their github and required only common reinforcement learning packages it was quick and painless to run. It was trivial to use the algorithms on different environments from different packages and collect their results for analysis.

What was difficult

One of the biggest difficulties was the time and resource consumption's of the experiments. Running each algorithm on each environment with a sufficient number of random seeds took the vast majority of the time. We had a total runtime of around 310 GPU hours (or 13 days). Time was our primary constraint and was the primary reason we did no investigate other environments. Simulator differences also proved to be somewhat challenging.

Communication with original authors

Our contact with the authors was limited to a discussion we had at their poster presentation at NeurIPS 2020.

1 Introduction

Deep reinforcement learning (RL) has achieved a great deal in the past decade. From mastering games such as Go [21], Dota 2 [5], StarCraft II [25], and Atari [4], to precision robotic control [2] and robotic movements [11]. However, none of these are solved environments and we are always looking to find better and faster algorithms. Here we strive to evaluate a novel approach presented in Pan et al. [19], the Softmax Deep Double Deterministic Policy Gradient (SD3).

SD3 presents an empirical and theoretical argument for the usage of the softmax operator in continuous control reinforcement learning tasks. It is standard practice to use the softmax operator for stochastic algorithms in discrete environments. However, with the recent successes of entropy maximizing RL [10], [11], [12], [13], [7], [26], there has been an increase of interest in the softmax operator for RL [3], [22]. SD3 continues this trend, utilizing the softmax operator to expand upon and gain improvements over TD3 [9] on the MuJoCo benchmark.

In this work, we attempt to replicate the results of Pan et al. [19] utilizing a variety of environments based in the open source PyBullet physics simulator. These environments include those from the original paper and in addition to some that were not present, to test SD3's ability to generalize to other problems. These test environments utilize the PyBullet physics simulator and are adapted for OpenAI Gym [6] via PyBullet-Gym and includes many similar environments from the MuJoCo benchmark. We use PyBullet over MuJoco to support efforts to make reinforcement learning more equitable [18], and we reject exclusionary MuJoCo usage and conduct all of our experiments exclusively on free and open source software. Note that all code and results will be available for a final copy (but are not presented here to preserve anonymity).

2 Preliminaries

2.1 Reinforcement Learning Background

Prior to getting evaluating the claims of the paper, a proper background is essential. Reinforcement learning is a field of machine learning in which an agent seeks to maximize a numerical reward signal from an environment [23]. RL is often formalized as a Markov Decision Process (MDP), defined by the tuple $\langle \mathcal{S}, \mathcal{A}, R, \gamma \rangle$. Here $\mathcal{S}$ represents the set of states, $\mathcal{A}$ the set of actions, R the reward and γ the reward discount between 0 and 1. It is common to also see a P in this tuple representing the probability of state transitions; however, our environments are not stochastic and therefore $P = 1$. The goal of an RL algorithm is to design (or learn) a policy π such that it maximizes the expected return (also called objective): $J = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)|\pi\right]$.

There are a number of techniques to design the policy π, but the dominant trend in contemporary RL is to use non-linear function approximators (i.e. deep neural networks) to learn value and policy functions. The focus of this work are actor-critic algorithms. In these systems an actor (or policy) network outputs the actions and is updated with a critic network that learns the value function. These policy functions can either be stochastic or deterministic, i.e. they can either output the mean and standard deviation of the policy distribution for an action to be chosen from, or output a single value for the action. We will largely be considering deterministic policy algorithms. The policy is network, parameterized by θ, is denoted as π_θ. This allows us to represent the objective function J, as an expectation, $J(\pi_\theta) = \int_{\mathcal{S}} r(s, \pi_\theta(s))ds = \mathbb{E}\left[R(s, \pi_\theta(s))\right]$, take the gradient to yield $\nabla J(\pi_\theta) = \int_{\mathcal{S}} \nabla\pi_\theta(s)\nabla Q(s, \pi_\theta(s))ds = \mathbb{E}\left[\nabla\pi_\theta(s)\nabla Q(s, \pi_\theta(s))\right]$, where $Q(s, a)$ indicates the expected return of taking action a in state s [20]. This is known as the deterministic policy gradient. There are a variety of techniques to optimize this function and we will outline how these techniques lead to SD3. For a visual outline of this see Figure 1.

DPG is essentially what is presented above, the policy is updated via the gradient above and the Q network is updated with the standard Bellman error: $\mathcal{L}(Q_\theta) = r_t + \gamma Q_\theta(s_{t+1}, \pi_\theta(s_{t+1})) - Q_\theta(s_t, a_t)$ [20]. DDPG improved upon DPG by adding noise to the policies to increase exploration, and by adding target networks for the policy and value networks [9]. TD3 attempts to address the well know overestimation problem in Q learning. Because $\mathbb{E}\left[max_a Q(s_t, a)\right] \geq max_a \mathbb{E}\left[Q(s_t, a)\right]$, Q function approximators often overestimate the Q value leading to convergence problems [16]. To address these overestimation errors, TD3 borrows from Double DQN [24] and uses two (hence the name 'twin') Q function approximators in addition to updating the policy network less frequently [9]. While TD3 does successfully address the overestimation problem, it introduces the new underestimation problem, something SD3 tries to combat [19]. Parallel to this are the entropy based methods: soft Q Learning [10] and soft actor-critic [12]. These methods maximize a different objective than presented above due to the addition of an entropy term: $J(\pi) = \sum_{t=0}^{T} \mathbb{E}\left[r(s_t, a_t) + \alpha\mathcal{H}(\pi(s_t))\right]$. This entropy objective (under optimal conditions) is functionally the same as the softmax operator.

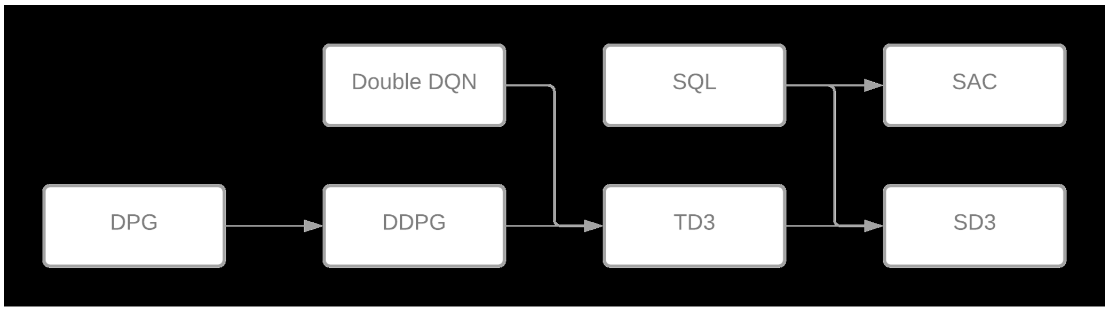


Figure 1: Outline of Continuous Control Algorithms. Deterministic Policy Gradient (DPG) [20], Deep Deterministic Policy Gradient (DDPG) [17], Double Deep Q Networks (Double DQN) [24], Twin Delayed Deep Deterministic Policy Gradient (TD3) [9], Soft Q Learning (SQL) [10], Softmax Deep Double Deterministic Policy Gradients (SD3) [19], Soft Actor-Critic (SAC) [13]

2.2 SD3

SD3 is essentially the same as TD3, with the main difference being the use of the softmax operator in the value function Bellman error. Hence, key to understanding SD3 is understanding the softmax operator. The softmax operator is common in RL problems, but is typically seen in discrete action spaces where it is easy to calculate: $\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=0}^{K} e^{z_j}}$. In continuous space action spaces, the form becomes computationally intractable: $\sigma(Q(s,\cdot)) = \int_{\mathcal{A}} \frac{exp(\beta Q(s,a))}{\int_{\mathcal{A}} exp(\beta Q(s,a'))da'} Q(s,a)da$. However, Pan et al. [19] proves helpful bounds on the difference between $max_q Q(s,a)$ and $\sigma\left(Q(s,a)\right)$, showing that the softmax operator does not overestimate and worst case only slightly underestimate. In order to make the continuous softmax operator computationally feasible, SD3 utilizes a sampling technique from Haarnoja et al. [13]: $\sigma(Q(s,\cdot)) = \mathbb{E}\left[exp(\beta Q(s,a))Q(s,a)\right] / \mathbb{E}\left[exp(\beta Q(s,a))\right]$. To illuminate the similarities and differences between the TD3 and SD3 we present them side by side highlighting a few key differences (blue are highlighted in both to show differences and red are only highlighted in one to show addition of a new feature).

```
Algorithm 1: TD3
Initialize value networks $Q_1, Q_2$ with
  parameters $\theta_1, \theta_2$
Initialize policy network $\pi$ with parameters $\phi$
Initialize target networks $Q'_1, Q'_2, \pi'$ with
  parameters $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{D}$
for for t = 0 to T do
    Select noisy action $a \leftarrow \pi(s) + \mathcal{N}$ and
      observe reward and new state $s'$
    Store $\langle s, a, r, s' \rangle$ in $\mathcal{D}$
    Randomly sample N tuples from $\mathcal{D}$
    $y \leftarrow r + \gamma min_{i=1,2} Q_{\theta'_i}(s', \pi_\phi(s') + \mathcal{N})$
    Update critics via the loss
      $\mathcal{L} \leftarrow \frac{1}{N} \sum (y - Q_{\theta_i}(s,a))^2$
    if policy update then
        Update $\phi$ via gradient
          $\frac{1}{N} \sum \nabla \pi_\phi(s) \nabla Q_{\theta_1}(s, \pi_\phi(s) + \mathcal{N})$
    if target update then
        $\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i$
        $\phi' \leftarrow \tau\phi + (1-\tau)\phi'$
```

```
Algorithm 2: SD3
Initialize value networks $Q_1, Q_2$ with parameters $\theta_1, \theta_2$
Initialize policy networks $\pi_1, \pi_2$ with parameters $\phi_1, \phi_2$
Initialize target networks $Q'_1, Q'_2, \pi'_1, \pi'_2$ with parameters
  $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi'_1 \leftarrow \phi_1$, $\phi'_2 \leftarrow \phi_2$
Initialize replay buffer $\mathcal{D}$
for for t = 0 to T do
    Select action $a \leftarrow \pi_i(s), i \leftarrow max_{i=1,2} Q_i(s, \pi_1(s))$
      and observe reward and new state $s'$
    Store $\langle s, a, r, s' \rangle$ in $\mathcal{D}$
    for i = 1, 2 do
        Randomly sample N tuples from $\mathcal{D}$
        Sample K noises $\epsilon$
        $\hat{a}' \leftarrow \pi_{\theta'_1}(s) + \epsilon$
        $\hat{Q} \leftarrow min_{i=1,2}(Q_{\theta'_i}(s', \hat{a}'))$
        $\sigma(\hat{Q}) \leftarrow exp(\beta\hat{Q}(s',\hat{a}'))\hat{Q}(s',\hat{a}')/exp(\beta\hat{Q}(s',\hat{a}'))$
        $y \leftarrow r + \gamma\sigma(\hat{Q})$
        Update $Q_{\theta_i}$ via the loss $\mathcal{L} = \frac{1}{N} \sum (y - Q_{\theta_i}(s,a))^2$
        Update $\phi_i$ via gradient
          $\frac{1}{N} \sum \nabla \pi_{\phi_i}(s) \nabla Q_{\theta_i}(s, \pi_{\phi_i}(s))$
    if target update then
        $\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i$
        $\phi'_i \leftarrow \tau\phi_i + (1-\tau)\phi'_i$
```

3 Methodology

3.1 Target Questions

In order to assess the validity of the paper and the conclusions it makes, we present three central questions.

- To what extent can we replicate the superior performance of SD3 over TD3 on the given environments?
- To what extent does this performance generalize to other continuous control tasks?
- What improvements can be made to the SD3 algorithm?

3.2 Experimental Setup

Although we provide TensorFlow [1] implementations of both TD3 and SD3, we run all experiments using the author's provided PyTorch implementations. This strengthens our results on generalizability, and is also more efficient due to code efficiency differences. As stated before, we opted not to use MuJoCo, favoring the open source pybullet which has the same environments as MuJoCo (in addition to many others). The PyBullet gym adaptions and implementations can be found here. We begin by collecting data for the 6 of the 8 environments in the original paper, specifically: Ant, Hopper, Lunar Lander, Walker2D, Humanoid, and Half Cheetah environments. We also evaluated three additional environments: Pendulum, InvertedDoublePendulum and HumanoidFlagrun. Similar to the original paper we collected data for 1 million iterations and repeated each experiment five times with a different random seed each time. For the extended experiments we only collected three runs due to time constraints. All experiments were conducted on two personal computers with CUDA enabled GPUs, specifically a GTX 1060 and a GTX 1080. Depending on the environment each run would take between 2 - 8 hours.

3.3 Hyperparameters

We largely used the same hyperparameters as the original paper. For all environments and algorithms refer to Table 1.

Batch size	100
Network architecture (policy and value)	(400,300)
ADAM[15] learning rate	$1 * 10^{-3}$
Replay buffer size	$1 * 10^{6}$
Training delay	$1 * 10^{4}$
Noise, $\mathcal{N}$	$\mathcal{N}(0, 0.1)$
γ	0.99
τ	0.005
Policy update frequency (TD3 only)	2
K	50

Table 1: Hyperparameters

The one notable difference is that Pan et al. [19] uses a separate set of hyperparameters for the Humanoid environment which we do not do. The second important note on hyperparameters is the SD3 unique hyperparameter β. In the original paper this is determined to be a specific value for each environment, ranging from 0.001 to 500. On the environments utilized in the paper we use the same values of β. Given our time and computational constraints, we do not perform ablation studies on the extended environments to determine the optimal β. merely adopting values from similar environments. For all β value see Table 2

4 Results

Our results are overall indicative that SD3 does provide an advantage on the some of the environments, although these advantages are relatively small.

4.1 Results on Paper Benchmarks

Results for six of the eight original environments can be seen in Figure 2. Blue represents SD3 and red TD3. The shaded area represents a confidence interval of one standard deviation. The exact numerical reward values of the

Ant	0.001
Half Cheetah	0.005
Hopper	0.05
Lunar Lander	0.5
Walker 2D	0.1
Humanoid	0.05
Pendulum	0.5
Inverted Double Pendulum	0.5
Humanoid Flagrun	0.05

Table 2: β Values

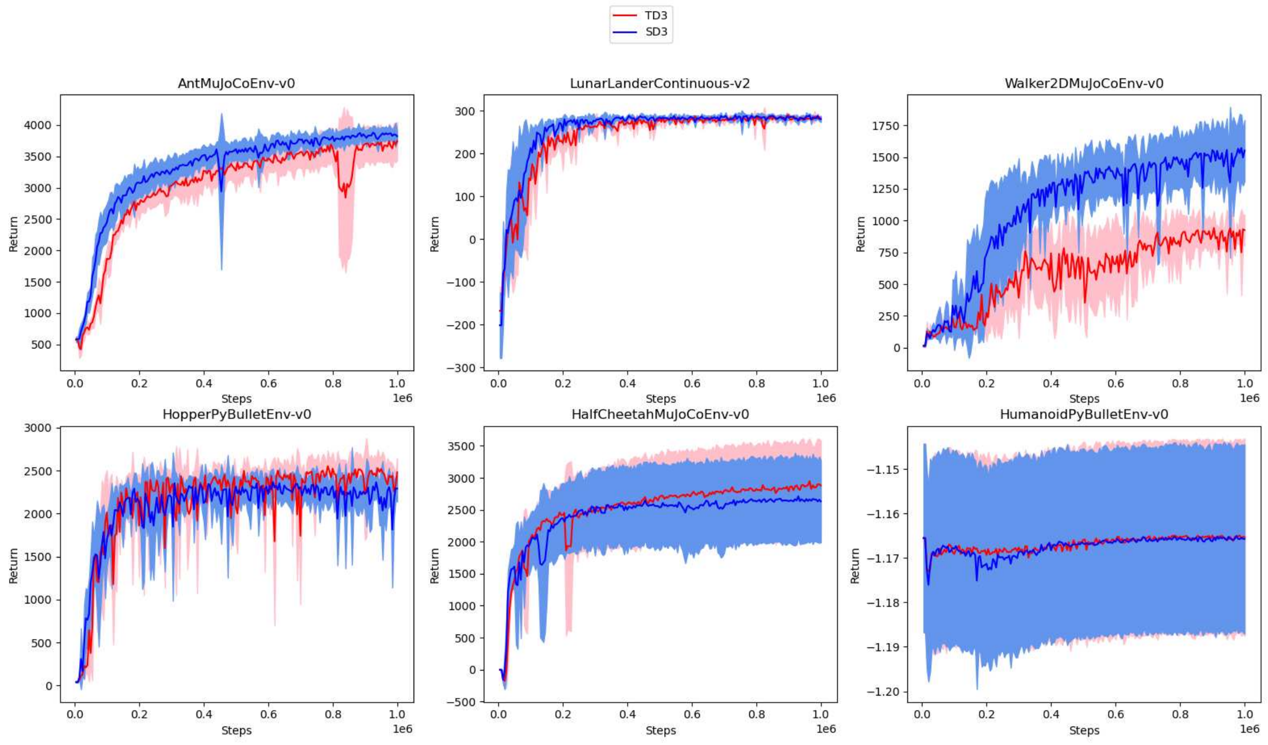
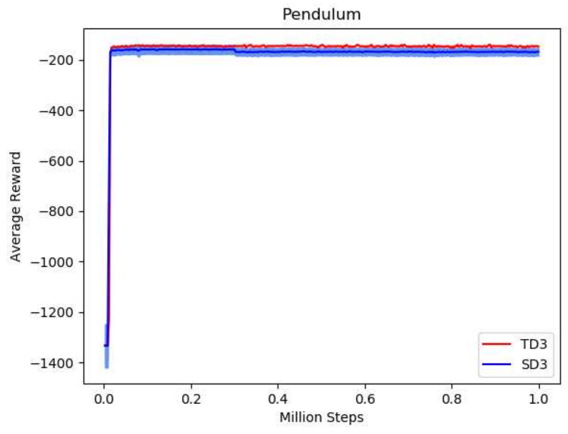


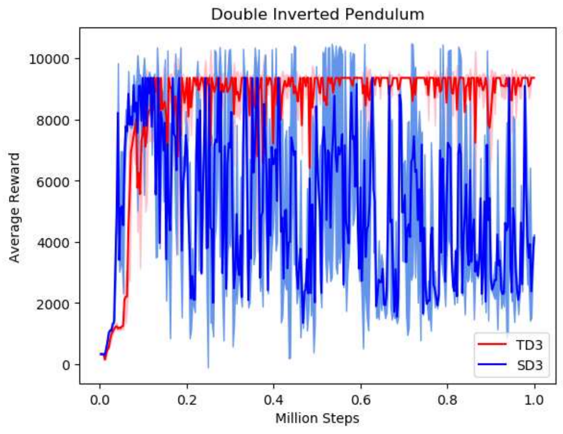
Figure 2: Paper Environments Reward vs Million Steps

PyBullet are not directly comparable to the MuJoCo rewards; the environments are similar but not exactly the same with PyBullet being supposedly harder. The key takeaway from these graphs is the relative performance of the two algorithms. SD3 presents slight improvements for Ant, comparable performance for Lunar Lander, Half Cheetah, and Hopper, and definitively superior performance on Walker2D. Walker2D is the only environment that one could universally recommend SD3 over TD3 as in every other environment the standard deviation curves overlap. There was a potential issue with the Humanoid PyBullet environment, which has seemed to have hardly any change for both algorithms.

4.2 Additional results not present in the original paper



(a) Pendulum Reward vs Million Steps



(b) Double Inverted Pendulum Reward vs Million Steps

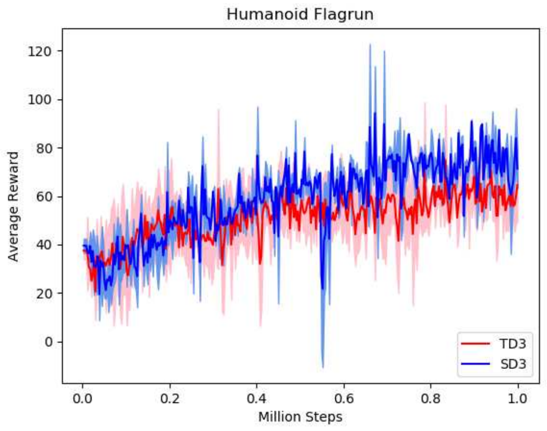


Figure 3: Humanoid Flagrun Reward vs Million Steps

Standard benchmarks are often problematic and may not always generalize (a reinforcement learning of "teaching for the test" so to speak). While there are proposals for other metrics of evaluation [14], we choose to simply evaluate the algorithm on environments that are not part of the "standard" benchmark. These three environments present similarly to the environmental results. Specifically on Pendulum and Humanoid Flagrun, SD3 offers slight advantages but on double inverted pendulum SD3 is far too volatile and performs inferior to TD3. We acknowledge that these may not be the optimal results for SD3, that being said, the β values are well within the standard range. These results are less about the potential strength of SD3 compared to TD3, and more about the generalizability and out of the box usability.

5 Discussion

First, let us consider the target questions we set in the beginning of this work. To what extent can we replicate the superior performance of SD3 over TD3 on the given environments? Somewhat. We were able to replicate generally good performance of SD3. However, the size of the advantages present in the original paper do not appear to be as large here. Pan et al. [19] presents 3 environments that SD3 definitively (i.e. the standard deviation curves do not overlap) performs better on: Half Cheetah, Ant, Walker2d, Hopper. However, we were only able to replicate this level of superior performance on Walker2d. On the other three environments, SD3's advantage was minimal to nonexistent. This is not to suggest that there is anything wrong with their results, rather, that the results may not generalize as well as hoped. Because PyBullet is not the same as MuJoCo (although the tasks are comparable) the exact comparison isn't necessarily important; however, the lack of clear advantages is.

To what extent does this performance generalize to other continuous control tasks? As was mentioned above, this generalization is weak. The generalization is minimal even to the same environments in a different physics simulator and this is also true for the new environments. We cannot say SD3 is the inferior algorithm on Pendulum and Double Inverted Pendulum as we did not do the full ablation studies to determine the optimal β values. However, we can say that if one wants SD3 to perform definitely better, specific values of β are needed, and even then performance may not be superior to TD3. This need for hyperparameter optimization is a weakness of the algorithm. Given the already numerous challenges of real world RL [8], requiring extensive trial and error to obtain the necessary parameters for algorithmic superiority is a steep price.

What improvement can be made to the SD3 algorithm? The most obvious improvement is to make the β parameter automatically adjustable. This idea is very similar to the improvements made to Soft-Actor critic. In the original paper, the entropy parameter, α, was determined via trial and error [12]; however, automating this parameter showed to be more effective from both a computation expense and a maximum reward standpoint [13].

6 Conclusion

In this work, we evaluate the reprehensibility of the paper Softmax Deep Double Deterministic Policy Gradients [19]. To promote inclusive research practices, we ran all code on the open source PyBullet physics engine. Our results generally align with the original paper's but are not very compelling overall. Some environments saw somewhat better performance with SD3, but many others saw similar performance with both SD3 and TD3. This may warrant further investigation there is the potential that hyperparameter optimization would bolster the performance of SD3. It is worth noting that this level of scrutiny is not applied to all algorithms, and we make no claims that other SotA continuous control algorithms would generalize any better.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[2] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[3] Kavosh Asadi and Michael L Littman. An alternative softmax operator for reinforcement learning. In *International Conference on Machine Learning*, pages 243–252. PMLR, 2017.

[4] Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Zhaohan Daniel Guo, and Charles Blundell. Agent57: Outperforming the atari human benchmark. In *International Conference on Machine Learning*, pages 507–517. PMLR, 2020.

[5] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

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

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

[8] Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901*, 2019.

[9] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1587–1596, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/fujimoto18a.html.

[10] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning*, pages 1352–1361. PMLR, 2017.

[11] Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. Learning to walk via deep reinforcement learning. *arXiv preprint arXiv:1812.11103*, 2018.

[12] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.

[13] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[14] Andrew Ilyas, Logan Engstrom, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. A closer look at deep policy gradients. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=ryxdEkHtPS.

[15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[16] Qingfeng Lan, Yangchen Pan, Alona Fyshe, and Martha White. Maxmin q-learning: Controlling the estimation bias of q-learning. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=Bkg0u3Etwr.

[17] 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. In *International Conference on Learning Representations*, 2016.

[18] Johan S Obando-Ceron and Pablo Samuel Castro. Revisiting rainbow: Promoting more insightful and inclusive deep reinforcement learning research. *arXiv preprint arXiv:2011.14826*, 2020.

[19] Ling Pan, Qingpeng Cai, and Longbo Huang. Softmax deep double deterministic policy gradients. In *Neural Information Processing System*, 2020.

[20] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. PMLR, 2014.

[21] 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.

[22] Zhao Song, Ron Parr, and Lawrence Carin. Revisiting the softmax bellman operator: New benefits and new perspective. In *International Conference on Machine Learning*, pages 5916–5925. PMLR, 2019.

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

[24] 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.

[25] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[26] Patrick Nadeem Ward, Ariella Smofsky, and Avishek Joey Bose. Improving exploration in soft-actor-critic with normalizing flows policies. *arXiv preprint arXiv:1906.02771*, 2019.