In Search of Optimal Exploration: An Attempt to Enhance Policy Optimization Algorithms for Gymnasium Leaderboard Position

Benjamin Birk Longet - belon18@student.sdu.dk
Thor Kamp Opstrup - thops19@student.sdu.dk

University of Southern Denmark, Campusvej 55, 5230 Odense M, Denmark

Abstract. A novel attempt has been made to optimize Proximal Policy Optimization (PPO) and Trust Region Policy Optimization(TRPO) with the introduction of a combination of Ornstein–Uhlenbeck action noise(OU) and state Entropy Maximization with Random Encoders(Re3). The primary goal has been to achieve a decent position on the gymnasium leaderboard for the BipedalWalker-v3[1] challenge. The parameters for the additional exploration strategies has been thoroughly tested. Some impact has been found, but no result showed statistical significance that the additions optimized the exploration. The project concludes with a satisfying position on the leaderboard.

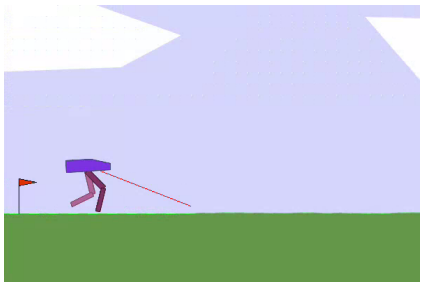
The code, data and sources can be examined at the GitHub repository developed for this project: *https://github.com/ThorKampOpstrup/Project-in-Artificial-Intelligence-gym-challenge*-[2]

1 Introduction

The reinforcement learning research field seeks to learn and be as efficient as possible in a wide range of applications. Numerous attempts have been made to optimize the learning for vastly different environments and tasks. This report will focus on achieving one goal, to get on the leaderboard for the BipedalWalker challenge[1]. This section covers the requirements of the challenge, followed by a brief explanation of the two core algorithms used.

1.1 The Bipedal Walker Challenge

OpenAI's Gymnasium consists of a numerous environments to train and test different reinforcement learning algorithms. Some of these environments includes a challenge where a certain fitness score needs to be achieved for it to be eligible for the leaderboard. The position on the leaderboard is based on the lowest amount of iterations the algorithm required to beat the challenge. The leaderboard serves as a benchmark to compare different algorithms.



(a) The BipedalWalker-v3 environment

User	Version	Episodes before solve
timurgepard	3.0	70 (Symphony🎹 ver 2.0)
timurgepard	3.0	100 (Monte-Carlo🌊 & Temporal Difference🔥)
Lauren	2.0	110
Mathias Åsberg 😎	2.0	164
liu	2.0	200 (AverageEpRet:338)

(b) Leaderboard of the BipedalWalker challenge as of 21/12/2023. The leaderboard is available **here**[1].

Fig. 1: Challenge

The observation space for the BipedalWalker (fig. 1a) contains 24 measurements and consists of the angle, angular velocity and positional velocity of the hull, angle and angular velocities of both hip and knee-joints, a boolean ground contact flag for each leg, and finally 10 lidar measurements for different angles from the hull to the environment. The action space consists of torque applied to the four joints. The model receives a reward for moving forward, a large penalty for falling, and a penalty for applying motor torque. The reward achievable can reach upwards 300+ at the end of the map[3]. The challenge for this environment is to obtain a mean reward of over 300 measured in 100 consecutive simulations[1].

The leaderboard as of 21/12/2023 is seen in fig. 1b. Most of the top solutions use Soft Actor-Critic or Twin Delayed Deep Deterministic Policy Gradient with fine-tuned hyperparameters. Only a few solutions use some novel algorithm, including the top two entries. While it is not explicitly specified what constitutes an episode, a reasonable assumption is that one iteration of the policy corresponds to one episode.

1.2 TRPO

Trust Region Policy Optimization[4] (TRPO) is an algorithm developed by OpenAI in 2015 as an improvement to the natural policy gradient method[5] which uses second order optimization. TRPO differs from natural policy gradient methods by incorporating a trust region constraint to the policy update, this constraint limits how much the policy is allowed to change during an update. The trust region is based on the current gradient, allowing larger steps when the gradient is flat, and smaller steps when the gradient is steep to ensure that the policy does not overshoot the optima. To increase performance compared to the natural policy gradient method, TRPO uses the conjugate gradient method, which is a numerical procedure that circumvents the need to compute the inverse of the Fisher matrix. The natural policy gradient presumes that the KL-constraint is met, which might not be true, therefore TRPO incorporates a backtracking line-search algorithm that iteratively reduces the trust region until the KL-constraint

is met, thereby enforcing it. As a final enhancement, TRPO runs an improvement check to verify that the surrogate advantage will improve.

1.3 PPO

Proximal Policy Optimization[6] (PPO) has some of the benefits of TRPO, while being much simpler to implement and tune, while occasionally having better sample complexity. Instead of constraining the update with trust regions, PPO utilizes a clipped surrogate objective function to limit the policy update to a range around the current policy. An adaptive KL penalty is employed in the objective function, this ensures that the updates become more conservative if the new policy deviates significantly from the previous one. PPO avoids the need for the conjugate gradient method used in TRPO, as it uses a simple stochastic gradient descent to find the new policy, which makes it more computationally efficient. When updating the policy, rather than doing one update per sample, PPO performs multiple epochs of updates, while dividing the data into mini-batches.

According to OpenAI;

> "*PPO has become the default reinforcement learning algorithm at OpenAI because of its ease of use and good performance*"[7]

2 Methods

PPO is implemented through Stable-Baselines3[8] (SB3) from OpenAI. TRPO is implemented with Stable-Baselines3-Contrib[9], which contains the experimental algorithms from OpenAI. These implementations has some additional optimizations compared to the vanilla versions from the papers. Some of these improvements include: Mini-batch updates, entropy regularization, gradient clipping, approximating the KL divergence and generalized advantage estimation.

2.1 State Entropy Maximization with Random Encoders

The main idea of State Entropy Maximization with Random Encoders[10] (Re3) is to estimate the entropy of the k-nearest neighbor of any given state in a single rollout. The entropy is estimated as the normalized and averaged distance between a state and the k-nearest neighbors. When maximizing entropy by including the distance as a secondary reward, the policy should promote diverse behaviors, which would hopefully result in more effective exploration.

To generalize the entropy measurement between states, a single randomly initialized encoder is used to encode the states into a representative state vector. According to the paper, the idea of using a random encoder is that two somewhat similar states would result in two somewhat similar encoded state representation and is therefore effective at capturing information about similarities between states. Re3 falls into the category of intrinsic reward modules, such

modules are most useful in sparse reward environments, as they guide the agent towards diverse behavior that might eventually obtain some extrinsic reward. Intrinsic reward modules occasionally show some promising results in regards to improving the exploration of reinforcement learning algorithms, this improvement is however very reliant on the environment and the underlying algorithm used.

The implementation of Re3 is based on the paper and inspired by the solution from the Reinforcement Learning Exploration Baselines repository[11]. The Re3 module is implemented with `pytorch`. It consists of two fully connected hidden layers, both with 64 neurons using ReLu activation. The final layer normalizes the output to a 128 dimensional feature vector, to represent the input information in a more complex manner. The gradient calculations for the network is deactivated to avoid unnecessary computations, as the weights should stay the same throughout the entire run.

2.2 Ornstein-Uhlenbeck Action Noise

Both of the tested algorithms use regular gaussian noise as exploration by adding randomness to the actions. In this project we try to apply a more complex type of noise, hoping that this might give some increase in performance. The Ornstein-Uhlenbeck process[12](OU) is used to generate temporarily correlated noise. In simple terms, it works by shifting the mean of a gaussian kernel over time based on the previous noise sample, much like a regular random-walk algorithm, along with a mean-reverting property. Mean reversion is essential in the OU process, as it ensures that the mean converges to, -or fluctuates around, a specific value instead of diverging towards infinity.

$$X_n = X_{n-1} + \theta(\mu - X_{n-1})dt + \sigma \Delta W_n \tag{1}$$

$$\Delta W_n = \sqrt{dt} N(0, 1) \tag{2}$$

According to [13], eq. (1) is a discretized approximate of the OU process, where ΔW_n are independent identically distributed Wiener increments, meaning gaussians with zero mean and Δt variance. ΔW_n can therefore be calculated by eq. (2). The mean-reversion factor is denoted by θ and determines the rate of which the mean of the noise converts back to μ. The noise added to each step is scaled with σ.

The effect of changing the parameters for a one-dimensional OU action noise is seen in fig. 2. A higher θ results in more gaussian-like noise, while a higher σ gives a more sporadic trajectory of the noise.

While recent work, [14] and [15], suggests that no performance is gained by using correlated noise, it is worth noting that these were applied on off-policy algorithms. Given that this project employs on-policy algorithms, OU noise is incorporated for exploration, as it might pose some advantage when exploring around the current policy.

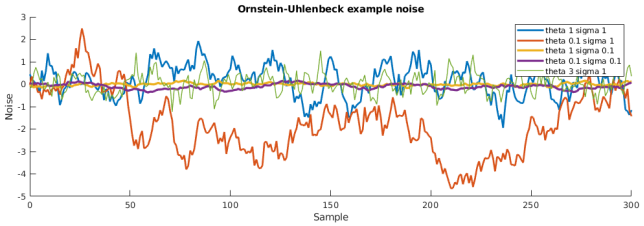


Fig. 2: Example of Ornstein-Uhlenbeck action noise with different parametres.

The implementation of OU noise is accomplished by adding the OU process directly to the actions when collecting the rollout data within the ActorCritic base class of SB3. The previous noise sample is reset at the beginning of every rollout as a precaution for the low θ tests. The noise is added on top of the existing gaussian noise implementation in an attempt to maintain the benefits of both noise sources.

2.3 Testing

Both algorithms are tested with hyperparameters close to the standard from SB3[8], the changes made are purely to optimize computation efficiency. The optimal hyperparameters for both algorithms are not found due to the excessive time it would take. The environment is simulated on 32 cores when collecting the rollouts, each environment collects 1024 steps, per core, per rollout, totalling 32768 datapoints. Each time a rollout is collected, a complete 100-mean evaluation is done based on the challenge requirement, before the policy is updated. A single run consists of 150 algorithm iterations. Tensorboard is used to log all data, and a log extractor script is created to convert from Tensorboard log files to CSV. The baselines for TRPO and PPO is gathered over 20 runs to achieve a solid mean value. Re3 and OU performance when testing for the optimal parameters are gathered over 5 runs per parameter change due to the computational constraint. A bar plot with both the best and the mean of the runs is used to decide which parameters seem optimal.

A total of 3 hyperparameters are tested. The k amount of nearest neighbors in Re3, the σ parameter for OU while using a θ value of 1, and finally a test for θ using the σ value from the previous test.

Finally, a comparison between the best results from the base algorithms and the results with the combined optimal additions will be made.

3 Results

The implementation, test scripts, log extractors plot generators, logs, models and extracted data can be examined in our **GitHub repository**[2].

3.1 Baseline test

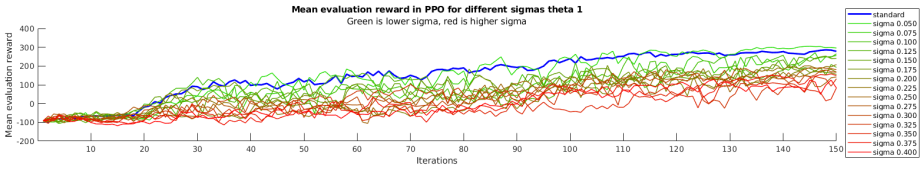
	PPO	TRPO
Mean	113.4	84.6
Best/lowest	58	58
Kolmogorov-Smirnov rejection	True	True
p-value	$2.78 * 10^{-17}$	$4.72 * 10^{-}19$

Table 1: Results of the standard PPO and TRPO tests, showing the amount of iterations needed to achieve an average score of 300, and the results of a Kolmogorov-Smirnov test.

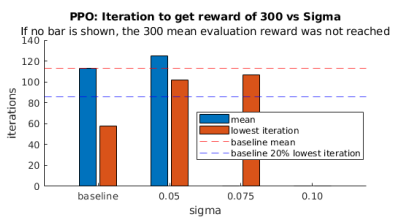
A Kolmogorov-Smirnov test for normality is applied to the baseline data to determine which statistical method are suited for the remaining tests.

3.2 Ornstein-Uhlenbeck σ-test

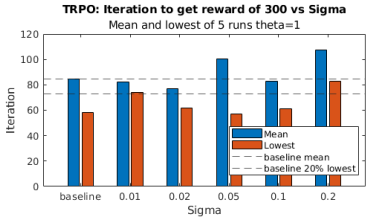
The tested values for σ are Sigma are tested in the range $[0.05, 0.40]$ for the PPO. Low σ is encoded with green and higher σ is encoded with red, blue is the baseline mean. Based on fig. 3a, TRPO are tested in the range $[0.01, 0.2]$. The TRPO result is excluded from the report as it contained limited informative content, but is available in the **GitHub repository**[16].



(a) Impact of sigma in OU noise on PPO



(b) PPO: Iteration to reach 300



(c) TRPO: Iteration to reach 300

Fig. 3: TRPO and PPO σ test

Based on fig. 3c the σ values of 0.01, 0.02 and 0.1 could be of interest for TRPO. The Kolmogorov-Smirnov seen in table 1 indicate that normality can

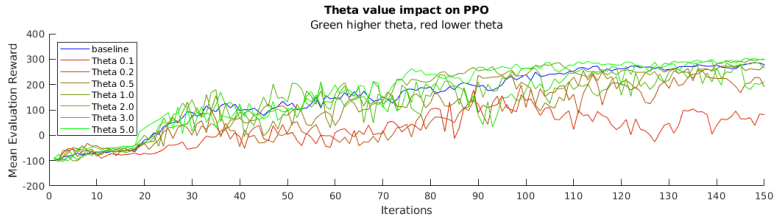
not be assumed. Equal variance is not assumed due to the limited sample size. The Mann-Whitney U test is used to test the null hypothesis that data is from continuous distributions with equal medians. A significance level of 5% has been selected. On table 2 a summation of the test can be seen.

Sigma	0.01	0.02	0.1
p-value	0.946	0.434	0.973
Mann-Whitney rejection	False	False	False

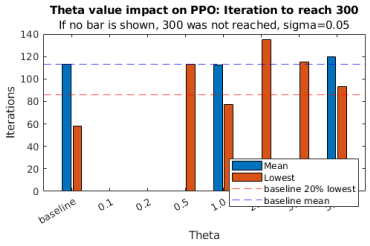
Table 2: TRPO: Results of Mann-Whitney U test of σ's of interest against baseline

3.3 Ornstein-Uhlenbeck θ-test

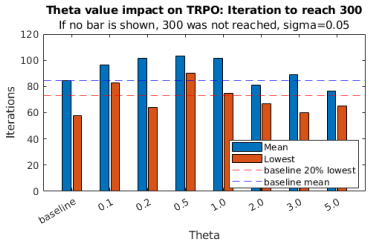
The θ value has been tested in the range $[0.01, 5.0]$, the impact on PPO mean evaluation reward can be seen in fig. 4a. The plot of TRPO is once again excluded, but can be examined on the GitHub repository[16].



(a) θ test on PPO



(b) PPO: Iteration to reach 300



(c) TRPO: Iteration to reach 300

Fig. 4: TRPO and PPO θ test

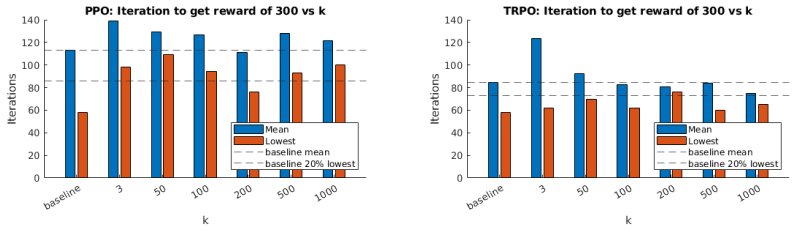
A Mann-Whitney U test has been conducted with the same hypothesis and significance level as in section section 3.2.

Theta	2.0	5.0
p-value	0.586	0.248
Mann-Whitney rejection	False	False

Table 3: TRPO: Results of Mann-Whitney U Test on θ of interest against baseline.

3.4 Re3 optimal k-test

The k amount of neighbors are tested in the range $[3, 1000]$. The resulting bar plots can be seen on fig. 5.



(a) PPO (b) TRPO

Fig. 5: Iteration to get reward of 300

The PPO does not seem to have any k value besides the k=100 that is close to or better than the baseline, thus no Mann-Whitney U test has been conducted for the addition of Re3 to PPO. A Mann-Whitney U test has been performed on the population using $k = 1000$ versus the baseline.

k	1000
p-value	0.153
Mann-Whitney rejection	False

Table 4: TRPO: Results of Mann-Whitney U test of k of interest against baseline

3.5 Combined Re3 and OU Action Noise test

A combination of both Re3 and OU action noise have been tested based on the previous tests. The selected parameters can be seen in table 5.

	PPO	TRPO
sigma	0.02	0.02
theta	1.0	5.0
k	200	1000

Table 5: Hyper parameters for the final combined test.

PPO's σ is determined based on the result from TRPO, and the indication that an even lower value than the one tested would be better. 10 runs are conducted for both algorithms with the added functionality. The average and best runs obtained are shown in fig. 6.

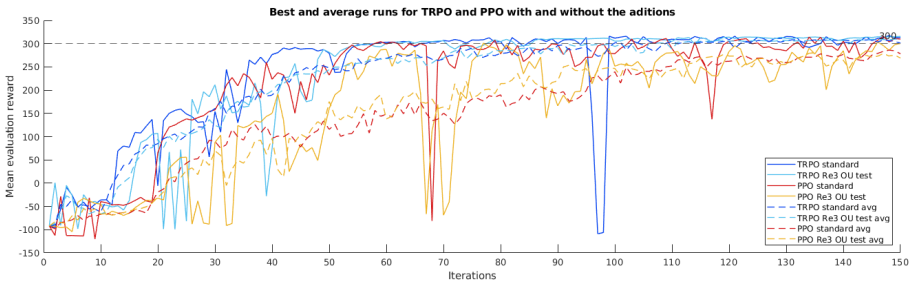


Fig. 6: Result of the final algorithms, showing both the mean and the best runs.

	PPO	TRPO
Mean	116.5*	87.3
Best/lowest	77	60
p-value	0.83*	0.53
Mann-Whitney rejection	False*	False

Table 6: Test statistics of PPO and TRPO with added Re3 and UO action noise against the baseline. **Some runs did not reach 300 and have been excluded for the statistics*

4 Analysis and Discussion

4.1 Ornstein-Uhlenbeck

The Mann–Whitney U test showed no statistical significance for any of the tested parameters, indicating no advantage was gained by the use of OU noise.

When looking at the tests using TRPO fig. 4b, a few parameters did show some indication of improvement, but not enough to establish that OU noise is better in any way.

All the final values used both had a high θ and a low σ, which effectively applies a very subtle gaussian noise, further proving that OU noise had no advantageous effect.

It could have been worth applying OU noise without the existing gaussian noise to see the actual effect that it has on exploration. This was not done as the focus was to improve the existing exploration.

4.2 Re3

The Mann-Whitney U test indicated no statistical significance. The p-value for the test using TRPO with k=1000 is somewhat low, which could indicate that more samples or fine-tuned parameters could result in an improvement to the algorithm.

The reason for the negligible effect of Re3 might have to do with the fact that single states are rarely that different from the rest, at least for the bipedal walker environment. Specifically designed reward functions that account for the limited differences between states might fare better in this environment. Re3 might be better suited for tasks that uses a larger state space such as images, as the actual differences might be more prominent.

When looking at the plots for mean Re3 reward, which is located in the tensorboard files, it is worth noting that on the rollouts where a high Re3 reward is achieved, the mean evaluation fitness behaves very sporadically. Whether the Re3 reward is the cause of the sporadic fitness, or the sporadic fitness is the cause of the high Re3 reward, it might indicate that Re3 does influence the exploration, usually in a negative manner[17].

Papers that focus on intrinsic rewards usually revolves around solving sparse reward environments. The limited amount of papers that claim to enhance the sample efficiency, often show very little improvements. This could be an indication that intrinsic reward methods, in general, does not fare well in regards of enhancing exploration in dense reward environments.

4.3 General

The initial expectations of both Re3 and OU noise applied to on-policy algorithms were not optimistic to begin with, as two papers claimed no advantage is gained with OU noise, and intrinsic rewards are mostly useful for solving sparse reward environments.

Given the limited sample size, a single lucky run, -or the lack of one, can have a great impact on the statistics for the tested hyperparameter. Additionally, the small sample size makes achieving statistical significance challenging in non-parametric tests, even if the actual performance might be favorable. The low sample size is due to the time required to optimize a policy (total computation time was around 307 hours or 12.8 days).

Another reason for the poor results might be due to the fact that the implemented algorithms had a lot of other optimizations already applied, meaning that one or two minor optimizations such as Re3 and OU noise might be unable to improve the algorithms further. A better way of testing this would have been to implement the algorithms without any additional methods, to see whether any improvement is gained for the core algorithms.

5 Conclusion

In conclusion, there has not been found a statistical advantage of the addition of Re3 and OU action noise for the bipedal walker challenge. Some indications of the potential for improvement TRPO in regards to both additions were found, though not enough to conclude that the additions were beneficial.

The results for the mean and best single runs for every test is seen in table 7. Pairing these results with the leaderboard fig. 1b, all best runs using TRPO, as well as regular PPO, achieved the first place. PPO with the addition of OU noise achieved a third place, and the rest achieved the second place.

	PPO	TRPO	PPO +OU	TRPO +OU	PPO +Re3	TRPO +Re3	PPO +OU +Re3	TRPO +OU +Re3
Runs	20	20	5	5	5	5	20	20
Hyper-parameters			$\sigma = 0.05$ $\theta = 1$	b(σ=0.5) m(σ=0.02) θ=1	k=200	b(k=500) m(k=1000)	k=200 σ=0.02 θ=1	k=1000 σ=0.02 θ=5
Lowest/best	58	58	102	57	76	60	77	60
Mean	113.39	84.60	124.8	77.2	111.4	74.6	116.5	87.25

Table 7: Summary of finds b = *best,* m = *mean*

A decent position on the leaderboard was acquired, though it was mostly due to the standard algorithms in conjunction with the high amount of samples gathered per rollout.

6 Contribution

1. Intellectual and analytic contributions (Benjamin 60%, Thor 40%)
2. Writing/implementation of code (Benjamin 70%, Thor 30%)
3. Production of experiment data (Benjamin 30%, Thor 70%)
4. Writing report (Benjamin 50%, Thor 50%)

References

1. OpenAI, "Gymnasium challenge leaderboard," [Online; accessed 21-12-2023]. [Online]. Available: https://github.com/openai/gym/wiki/Leaderboard
2. B. B. Longet and T. K. Opstrup, "Github," [Online; accessed 8-01-2024]. [Online]. Available: https://github.com/ThorKampOpstrup/Project-in-Artificial-Intelligence-gym-challenge-
3. O. Klimov, "Bipedal walker," [Online; accessed 9-01-2024]. [Online]. Available: https://gymnasium.farama.org/environments/box2d/bipedal_walker/
4. J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*. PMLR, 2015, pp. 1889–1897.
5. S. M. Kakade, "A natural policy gradient," *Advances in neural information processing systems*, vol. 14, 2001.
6. J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.
7. OpenAI, "Openai's introduction to ppo," [Online; accessed 04-01-2024]. [Online]. Available: https://openai.com/research/openai-baselines-ppo
8. A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html
9. OpenAI, "Stable-baselines3-contrib," [Online; accessed 9-01-2024]. [Online]. Available: https://github.com/Stable-Baselines-Team/stable-baselines3-contrib
10. Y. Seo, L. Chen, J. Shin, H. Lee, P. Abbeel, and K. Lee, "State entropy maximization with random encoders for efficient exploration," in *International Conference on Machine Learning*. PMLR, 2021, pp. 9443–9454.
11. F. Yuan, "Reinforcement learning exploration baselines (rlexplore)," [Online; accessed 04-01-2024]. [Online]. Available: https://github.com/yuanmingqi/rl-exploration-baselines/tree/main
12. Wikipedia, "Ornstein–uhlenbeck process," [Online; accessed 04-01-2024]. [Online]. Available: https://en.wikipedia.org/wiki/Ornstein%E2%80%93Uhlenbeck_process
13. math.stackexchange.com, "Ornstein–uhlenbeck process, discretized," [Online; accessed 13-12-2023]. [Online]. Available: http://math.stackexchange.com/questions/1287634/implementing-ornstein-uhlenbeck-in-matlab
14. S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International conference on machine learning*. PMLR, 2018, pp. 1587–1596.
15. G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, D. Tb, A. Muldal, N. Heess, and T. Lillicrap, "Distributed distributional deterministic policy gradients," *arXiv preprint arXiv:1804.08617*, 2018.
16. B. B. Longet and T. K. Opstrup, "Github - figures," [Online; accessed 8-01-2024]. [Online]. Available: https://github.com/ThorKampOpstrup/Project-in-Artificial-Intelligence-gym-challenge-/blob/main/plotter/plots/README.md
17. ——, "Github - re3 reward example," [Online; accessed 8-01-2024]. [Online]. Available: https://github.com/ThorKampOpstrup/Project-in-Artificial-Intelligence-gym-challenge-/blob/main/plotter/plots/Re3_ex/README.md