An Improved and Simplified Method for Reinforcement Learning

Anonymous Authors[1]

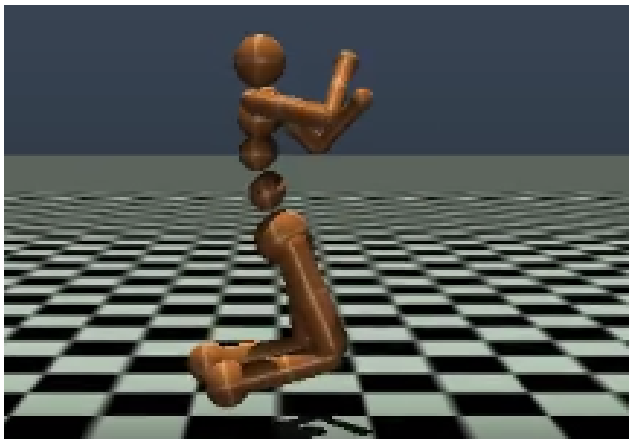


Figure 1. Gymnasium MuJoCo's Humanoid-v4. "Teaching" this figure to walk is a common Reinforcement Learning benchmark.

Abstract

This paper studies the recent advances and issues in the field of continuous, off-policy Reinforcement Learning algorithms. Particular attention is given to policy normalization techniques, the Twin Delayed Deterministic Policy Gradient Algorithm and the Streamlined Off-Policy Algorithm. Finally, the SOP algorithm is analyzed and re-implemented. This paper replicates the analyses of (Che et al., 2020) and demonstrates SOP to be a successful solution to many continuous RL tasks.

Implementation Paper

1. Introduction

Reinforcement Learning has proven to be a powerful tool in a number of domains. However, one of the most fascinating applications is teaching robots to move like living creatures. As any parent knows, teaching a child to walk is a difficult, but rewarding task. Human's enjoy the evolutionary benefit of naturally learning from those around them. However, Robots are not as lucky. Reinforcement Learning, especially in continuous domains such as robotics, is a perilous task. In this paper, I explain the hurdles that makes learning in continuous domains so difficult (Ahmed et al., 2019). I then discuss historical solutions proposed to solve said issue(Fujimoto et al., 2018). Finally, I analyze and re-implement the Streamlined Off-Policy Algorithm (Che et al., 2020). I reiterate that it is capable of learning in a wide variety of continuous scenarios (including some previously untested enviroments). Finally, I analyze the Normalization proposed by (Che et al., 2020) and discuss where it succeeds and fails.

2. Background: An incredibly brief introduction to RL

Reinforcement Learning distinguishes itself from other Machine Learning methods by not requiring large static datasets. Instead, Reinforcement Models, aka policies (π), are trained on a series of states (s), actions (a), and corresponding rewards (r) gathered from their interactions with an environment. The goal is to find an optimal policy (π^*) that follows trajectories (τ) that generate the maximum cumulative reward $R(\tau)$. Trajectories, aka episodes, are sequences of states and actions. Formally, the goal of Reinforcement learning is:

$$\pi^* = argmax J(\pi) \tag{1}$$

where

$$J(\pi) = \int_{\tau} \mathbb{P}(\tau|\pi) R(\tau) = \mathbb{E}_{\tau\sim\pi}[R(\tau)] \tag{2}$$

$\mathbb{P}(\tau|\pi)$ is the likelihood of the episode τ when following the policy π.

This paper primarily focuses on off-policy, Q-Learning RL algorithms. On-policy algorithms utilize the same policy to update π and gather data from the environment. In off-policy algorithms, π is used to gather data from the environment, but a different policy (usually an older version of π) is used to update π.

Q-learning is a subset of off-policy algorithms that learn an approximation of the enviroment's Q-function. The Action-Value function, $Q^\pi(s, a)$, returns the expected total reward

if action a is taken in state s and π is permanently followed after. Actor-critic algorithms, in which Q (the critic) and pi (the actor) are represented by Neural Networks, are a very popular variant of Q-Learning.

3. Understanding the Impact of Entropy on Policy Optimization

3.1. Problem

Neural Networks exist to solve problems that Humans cannot create formal algorithms for. This is simultaneously the greatest strength and weakness of Deep Learning, as researchers are severely lacking in their ability to deeply understand the training of such networks. Methods that make the "Black Box" of Neural Network training more transparent are highly desirable for many reasons. Such methods help identify deficiencies in networks and clarify the direction of future objectives or architectures. Reinforcement Learning, especially the actor-critic algorithms discussed in this paper, are famously unstable (Che & Ross, 2019) and stand much to gain from these insights.

3.2. Prior Work

Linear Interpolations (Chapelle & Wu, 2009) have been a historically popular way to represent the "Geometry" of an objective function O. Such methods are achieved by observing the 1D subspace between parameter states θ_0 and θ_1. $O((1-\alpha)\theta_0 + \alpha\theta_1)$ is then graphed from $\alpha = 0$ to $\alpha = 1$.

3.3. Research Gap

This method often provides good visualizations, however they are limited to the 1D slice between θ_0 and θ_1. This means making generalizations from such interpolations is difficult given that the optimization may proceed in infinitely different directions.

3.4. Contributions

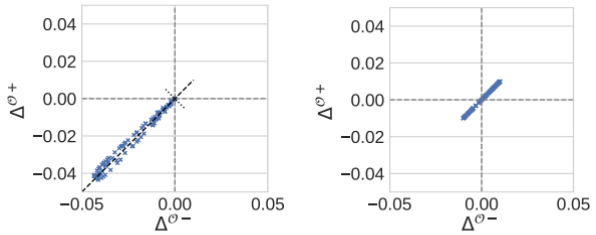
(Ahmed et al., 2019) introduces the Random Perturbation Visualization algorithm. This algorithm is shown to be capable of rigorously evaluating simple and complex objective functions. Finally the insights provided by the algorithm are used to demonstrate that objective function geography is correlated to the success of entropy terms in actor critic settings.

3.5. Proposed Solution

(Ahmed et al., 2019) proposes "Random Perturbation Visualization", a novel way to evaluate objective function geography. Instead of Visualizing in just one direction θ_1, a batch of directions d are sampled at random from the unit circle. Each direction is then evaluated forward and backward $\theta_d^+ = \theta_0 + \alpha d$, $\theta_d^- = \theta_0 - \alpha d$. Finally, the changes in objective O are evaluated across $\Delta_d^{O^-} = O(\theta_d^-) - O(\theta_0)$ and $\Delta_d^{O^+} = O(\theta_d^+) - O(\theta_0)$. This reveals a host of useful information. For example, if it is found that $\Delta_d^{O^-} < 0$ and $\Delta_d^{O^+} < 0$ for all d, we can say with significant confidence that θ_0 is a local maximum.

3.6. Claim and Evidence 1

(Ahmed et al., 2019) first sets out to prove that it's visualization method can Identify simple features such as local optimums and saddle points. A simple objective $O(\theta) = -(1 - \theta_0\theta_1)^2$ is trained to a local optimum at point at $\theta = (-0.5, -2)$ and a saddle point at $\theta = (0, 0)$. The results in Figure 2, demonstrate that the Random perturbation method is capable of representing both.



(a) Apparent local optimum (b) Saddle point

Figure 2. Two scatter plot of all $\Delta_d^{O^+}$ and $\Delta_d^{O^-}$ at two θ_0. (a) Since all points are negative, this indicates a local optimum. (b) Since all points are either both positive or both negative this indicates a saddle point.

3.7. Claim and Evidence 2

Now that the algorithm has been proven valuable in a basic scenario, (Ahmed et al., 2019) seeks to rigorously evaluate Perturbation Visualization in the RL domain. In this experiment actor critic objectives with varying levels of entropy are tested on the MuJoCo Hopper benchmark. The curvature of a given direction can be inferred by assuming that O is locally quadratic ($O(\theta) = a^T\theta + \frac{1}{2}\theta^T H\theta$) at θ_0. With this assumption it is possible to derive: $\Delta_d^{O^+} - \Delta_d^{O^-} = 2\alpha\nabla O(\theta_0)^T d$ and $\Delta_d^{O^+} + \Delta_d^{O^-} = \alpha^2 d^T H d$. Mapping the curvature of the aforementioned actor critic objectives provides us with the plot in Figure 4. This demonstrates that the Random Perturbation algorithm is capable of providing powerful insights into complex optimizations.

3.8. Claim and Evidence 3

Finally, (Ahmed et al., 2019) seeks to act on the insights provided by Figure 3. In this final experiment 3 MuJoCo benchmarks are trained using actor critic models with varying entropy coefficients σ. Figure 4, demonstrates that the

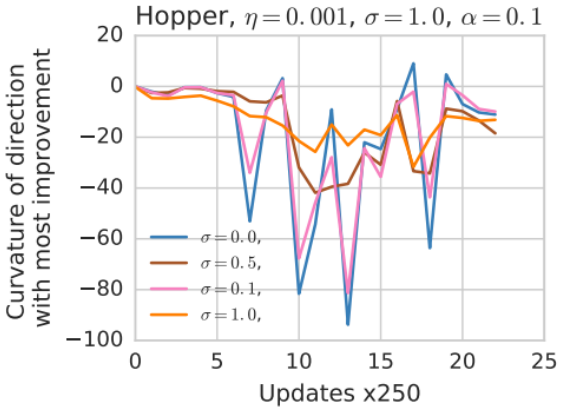


Figure 3. A visualization of the Geography of RL objective functions with differing levels of entropy. Notice that higher entropy levels lead to smoother geographies. This information provides (Ahmed et al., 2019) with a Hypothesis that is proven in Section 3.8

.

high entropy models consistently train faster and generally achieve higher cumulative awards. This is likely due the less jagged objective function created by the entropy term, as seen in Figure 3.

3.9. Critique

Random Function perturbation is a significant advance in the ability to visualize objective functions during training. However, it is not perfect. As mentioned earlier, objective functions can be optimized in infinite, and typically very distinct, directions. Since Random Function perturbation can only sample a large, but finite, subset of these directions, it's visualization can theoretically be misleading. Overall, I find this paper proposes a very creative solution in a concise manner.

4. TD3: Addressing Function Approximation Error in Actor-Critic Methods

*Note: This paper is from ICML 2018, but was approved by Professor Inouye.

4.1. Problem

Reinforcement Learning is an incredibly powerful tool that has proven useful in fields such as Robotics, Recommendation Systems, and even the optimization of LLMs (Sun, 2023). Early iterations of Reinforcement Learning assume discrete action spaces. For example, this could mean representing the movement of a car by the states "acceleration on"

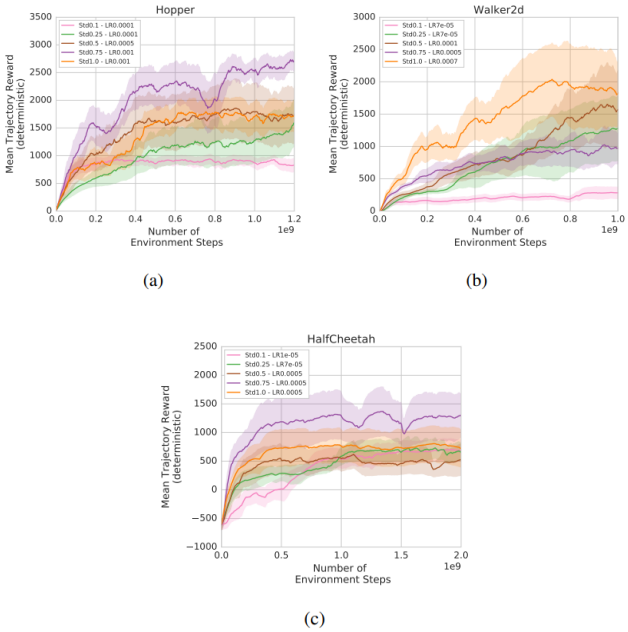


Figure 4. Cumulative award across 3 MuJoCo Benchmarks using different training techniques. σ represents the entropy coefficient.

and "acceleration off". However, we live in a continuous world and continuous action spaces are highly desirable. Moving from discrete space to continuous space introduces (and exacerbates) many issues, some of which are solved by the TD3 algorithm proposed by (Fujimoto et al., 2018).

4.2. Prior Work

Q-learning (as discussed above) was first proposed by (Watkins, 1989), which used a tabular method to approximate Q. Neural Networks were first used to estimate Q by (Mnih et al., 2013). However, neither method was able to successfully learn in continuous action spaces. Learning Q effectively in continuous action spaces is the primary contribution of the DDPG algorithm (Lillicrap et al., 2015), which this paper directly builds off of.

4.3. Research Gap

The DDPG algorithm suffers greatly from overestimation bias. In Q-learning the "target" of the Q function is estimated as $y = r + \gamma max_{a'} Q(s', a')$. If there is a positive error within Q, the maximum function will lead to an overestimation of the target. The Q network learns by minimizing the Mean Squared Bellman Error (MSBE), $\frac{1}{N}\sum^{N}(target - Q_{\theta_i}(s, a))^2$. Hence, this overestimation will recursively affect the learning of the Q network. Before

(Fujimoto et al., 2018), This overestimation bias was understood to exist in discrete Q Learning. However, its effects were unclear in continuous spaces.

4.4. Contributions

(Fujimoto et al., 2018) proves that overestimation bias severely handicaps the DDPG algorithm. To remedy the effects of this overestimation bias, (Fujimoto et al., 2018) Proposes the Twin Delayed Deterministic policy gradient algorithm (TD3). Finally, TD3 is demonstrated to achieve state-of-the-art results on multiple OpenAI Gym benchmarks.

4.5. Proposed Solution

The TD3 Algorithm utilizes two networks instead of 1 to represent the Q function. The lowest output of the two is used to compute the target, which reduces overestimation bias. Additionally, duplicate target networks are created for both the policy and Q networks. These target networks are used to generate the MSBE targets. Ultimately, these target networks make the algorithm more stable by not having the target and observation of MSBE as dependent on the same parameters. Finally, noise is added to $\tilde{a}$. This effectively regularizes the algorithm by evaluating nearby actions, even if π has a strong preference for a certain action value.

Algorithm 1 TD3

Initialize Twin Q Networks $Q_{\theta_1}, Q_{\theta_2}$
Initialize Policy Network π_ϕ
Initialize Target Networks $\theta_i' \leftarrow \theta_i$, $\phi_i' \leftarrow \phi_i$
Initialize replay buffer B
Initialize an environment Env with initial state s

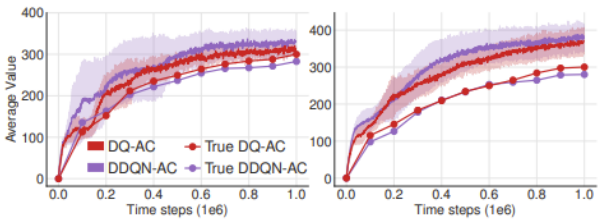
for $t \leftarrow 1$ to StepsToTrain **do**
  $\epsilon \sim N(0, \sigma)$
  $a \leftarrow \pi_\phi(s) + \epsilon$ ▷ Noise is added for exploration
  $s', r \leftarrow Env(a)$
  Add (s, a, r, s') to B

  Sample batch of N transitions from B
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \epsilon \sim \text{clip}(N(0, \sigma), -c, c)$
  $target \leftarrow r + \gamma min_{i=1,2} Q_{\theta_i'}(s', \tilde{a})$
  Update $\theta_i \leftarrow argmin_{\theta_i} N^{-1} \sum (target - Q_{\theta_i}(s, a))^2$

  if t mod d **then**
    Update ϕ with the policy gradient:
    $\nabla_\phi (N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s))$
    Update the Target Networks:
    $\theta_i' \leftarrow \tau\theta_i + (1 - \tau)\theta_i'$
    $\phi_i' \leftarrow \tau\phi_i + (1 - \tau)\phi_i'$
  end if
end for

4.6. Claim and Evidence 1

First, (Fujimoto et al., 2018) seeks to prove that overestimation bias exists in continous Q-Learning. This is done by testing the Double Q-Learning (DQN) and Double DQN (DDQN) algorithms on two MuJoCo environments. At each update, the actual Q(s,a) is calculated and compared to the model's Q.



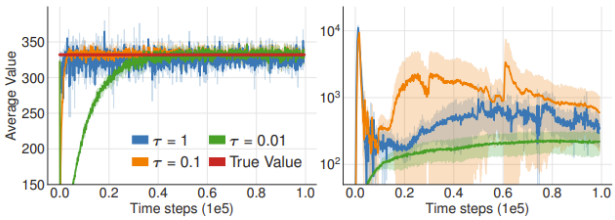
(a) Hopper-v1 (b) Walker2d-v1

Figure 5. Results of the experiment described in Section 4.6

Figure 4.6 demonstrates that overestimate bias will occur in continuous DQN algorithms unless precautions are taken.

4.7. Claim and Evidence 2

Additionally, (Fujimoto et al., 2018) seeks to prove the value brought to TD3 by introducing target networks. Hypothetically, using a slowly, instead of directly, updated network will improve training stability. In the following experiment, four Q-networks are trained with different update rates (τ). Their outputs are plotted against a fixed and learning policy network with the same update rate. (Fujimoto et al., 2018) argues that Figure 6, indicates that greater instability in networks without target networks may be the result of high variance in the Value function.



(a) Fixed Policy (b) Learned Policy

Figure 6. Value function outputs across different update rates τ. It is worth noting here that the goal is not to achieve the highest output of the Value function (as this could indicate an exploitative policy, or poor Q function), but rather a smooth optimization of the Value Function.

4.8. Claim and Evidence 3

Environment	TD3	DDPG
HalfCheetah	**9636.95 ± 859.065**	3305.60
Hopper	**3564.07 ± 114.74**	2020.46
Walker2d	**4682.82 ± 539.64**	1843.85
Ant	**4372.44 ± 1000.33**	1005.30
Reacher	**-3.60 ± 0.56**	-6.51
InvPendulum	**1000.00 ± 0.00**	**1000.00**
InvDoublePendulum	**9337.47 ± 14.96**	**9355.52**

Figure 7. TD3 and DDPG's maximum reward on various V1 MuJoCo Environments (as published by the TD3 paper)

At the time of publication, DDPG was state-of-the-art on MuJoCo Benchmarks. As Figure 7 demonstrates, TD3 vastly outperforms DDPG in almost all areas. Hence, the adaptations in section 4.5 marked a significant improvement relative to DDPG.

4.9. Critique

I found the experiments relating to Overestimation bias in this paper to be very clear and well-explained. Additionally, I found the method of taking the minimum of 2 Q networks to be a very elegant solutions. However, the claim made in Section 4.7 to incredibly unclear. In particular, plot (b) of Figure 6 appears to indicate that a medium speed network update has the greatest value variance of all. This would seem to contradict their claim that stability is encouraged by target networks.

5. Striving for simplicity and performance in off-policy DRL: Output Normalization and Non-Uniform Sampling

5.1. Problem

Reinforcement learning is notoriously difficult to train. This has only been exacerbated by the increasingly complex solutions that set the State of the Art. Complexity is often necessary to achieve better results; however, at a certain point algorithms must be practical to implement. Even more importantly, they must prove that all added complexity is necessary and not an extra component waiting to fail.

5.2. Prior Work

Deep Q Learning has been a popular RL method since "Playing Atari with Deep Reinforcement Learning" by (Mnih et al., 2013). DQN was extended into continuous action space with the DDPG algorithm by (Lillicrap et al., 2015). In 2018, The stability and effectiveness of DDPG was improved by the TD3 (Fujimoto et al., 2018) and Soft Actor-Critic (SAC) (Haarnoja et al., 2018) algorithms, which were published months apart.

5.3. Research Gap

The original iterations of TD3 and SAC proposed large changes to the already complex DDPG algorithm. Moreover, the value provided by certain additions, such as TD3's OU Noise or SAC's Value Network, was not clear. When dealing with multiple (often unstable) networks that are being trained simultaneously, adding complexity unnecessarily should be avoided at all costs. Additionally, Both algorithms required a Non-Uniform, Priority Experience Replay Buffer that required 100s of lines to implement.

5.4. Contributions

(Che et al., 2020) introduces the Streamlined Off-Policy Algorithm (SOP) and the Streamlined Off-Policy with Emphasized Recent Experience Algorithm (SOP_ERE). SOP drastically simplifies multiple aspects of the SAC and TD3 algorithms, while achieving comparable results on MuJoCo Enviroments. SOP_ERE is SOP but with a novel, simplified buffer scheme. SOP_ERE is shown to achieve state of the art results.

5.5. Proposed Solution

The complete SOP algorithm is provided in Algorithm 2. It simplifies TD3 by removing the target policy network π'_ϕ altogether. SAC is also simplified by the removal of the Value Network.

It's most notable contribution is it's simplified G Normalization scheme. Let $\pi = (\pi_1 ... \pi_K)$ be an output from the policy. Let $G = \sum_k \frac{\pi_k}{K}$ be the average magnitude of said output. Whenever $G > 1$ we set $\pi_k \leftarrow \pi_k / G$ for all k. This ensures that the average magnitude of a policy output is never greater than 1.

While not shown in Algorithm 2, the Emphasizing Recent Experience (ERE) buffer is also quite simple. The basic idea is to sample increasingly recent transitions (s, a, r, s') as time progresses. Let N bet the max size of the buffer, k be the update of the model and $\eta \in (0, 1]$ be a hyperparameter. ERE works by sampling uniformly from the c_k most recent data points, where $c_k = N\eta^k$. As k increases, c_k slowly narrows the window from which ERE may sample, thus emphasizing recent data.

5.6. Claim and Evidence 1

(Che et al., 2020) begins by identifying what they call the Squashing Exploration Problem. They start by observing that exploitative policy often explode outputs such that

Algorithm 2 Streamlined Off-Policy

Initialize Twin Q Networks $Q_{\theta_1}, Q_{\theta_2}$
Initialize Q Target Networks $\theta'_i \leftarrow \theta_i$
Initialize Policy Network π_ϕ
Initialize replay buffer B
At all times, if $G > 1$, normalize $\pi_\phi(s)$
while Not Converged **do**
  generate an episode of J actions using:
  $a = tanh(\pi_\phi(s) + \epsilon)$ where $\epsilon \sim N(0, \sigma)$
  for j in range(J) **do**

    get a batch $\beta \sim (s, a, r, s')$ from B
    compute targets for Q Functions: $\triangleright \epsilon \sim N(0, \sigma)$
    $y_q(r, s')$=$r + \gamma min_i Q_{\theta'_i}(s', Mtanh(\pi_\phi(s') + \epsilon))$
    Update the Q functions with MSBE:
    $\nabla_{\theta_i} \frac{1}{|\beta|} \sum_\beta (Q_{\theta_i}(s, a) - y_q(r, s'))^2$ for i=1,2
    Update policy by maximizing Q_{θ_1}:
    $\nabla_\phi \frac{1}{|\beta|} \sum_{s \in \beta} Q_{\theta_1}(s, Mtanh(\pi_\phi(s)))$
    Soft Update the target networks as in TD3:
    $\theta'_i \leftarrow \rho\theta'_i + (1 - \rho)\theta'_i$
  end for

$\pi_k(s) >> 1$ from $\pi(s) = (\pi_1(s)...\pi_K(s))$. Additionally, they note that in such situations $|\epsilon_k| << \pi_k(s)$. The end result is that actual action outputs are consitently pushed to the action bound $+M/-M$ because $Mtanh(\pi_k(s) + \epsilon_k) \approx Mtanh(\pi_k(s)) = M$. To demonstrate this, (Che et al., 2020) trains an SAC model with and without entropy loss on MuJoCo's Humanoid. The results in figure 8 confirms that without regularization of $\pi_k(s)$ itself, exploration is susceptible to "squashing".

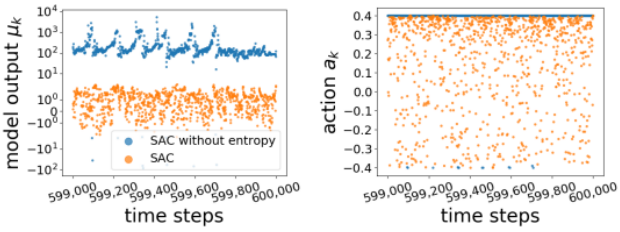


Figure 8. Left: π_k (called μ_k in the original paper) during training with and without entropy loss. Right: The corresponding $Mtanh(\pi_k(s) + \epsilon_k)$ outputs. Notice that without entropy, they are almost all equivalent to the action bound M

5.7. Claim and Evidence 2

In order to demonstrate the effectiveness of the SOP algorithm, it is compared SAC and TD3 on 5 MuJoCo Benchmarks. Additionally, it is compared to a modified version of itself, SOP_IG, where gradients are decreased if action outputs are too large. The results in Figure 9 demonstrate that both SOP and SOP_IG are capable of achieving comparable results to SAC and TD3.

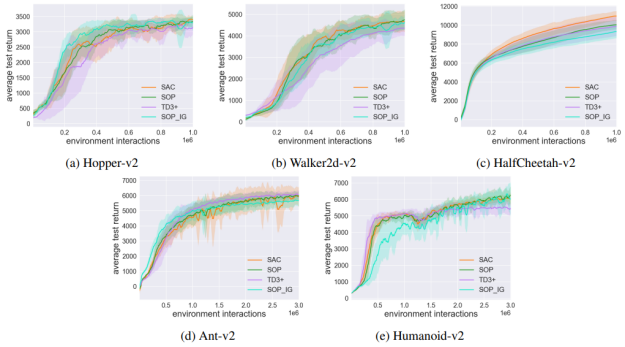


Figure 9. SOP and SOP_IG compared to TD3 and SAC on 5 Mujoco benchmarks

5.8. Claim and Evidence 3

Finally, (Che et al., 2020) demonstrates that SAC, SOP and SOP_IG are all capable of to state of the art results when combined with the ERE buffer. Performance is analyzed on the same 5 Mujoco benchmarks. The results in Figure 10 demonstrate ERE improves training performance.

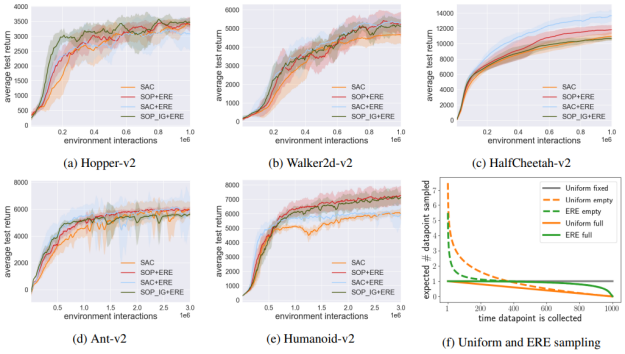


Figure 10. SAC+PER vs SAC+ERE vs SOP+ERE vs SOP_IG+ERE on 5 MuJoCo Benchmarks

5.9. Critique

As I will demonstrate in my implementation, the proposed normalization method completely fails in environments where the action space is one dimensional. This is not brought up by the Authors, but is obvious when looking

at their normalization function. Nonetheless, it exemplary and highly commendable that this paper achieves SOTA by *reducing* complexity.

6. Implementation Motivation

My primary goal is re-implement an SOP algorithm that can match (or come close to) the performance of (Che et al., 2020). This would mean training agents which achieve similar rewards on the MuJoCo environment, with the Humanoid being my highest objective. Beyond this I hope to make two small extensions. First, I would like to train successfully on at least one MuJoCo Environment that was not covered in the original SOP paper. Secondly, I would like to gain a deeper understanding of how Normalization affects in Reinforcement Learning tasks.

7. Implementation Plan and Setup

During Implementation, I chose to design the basic components first. This meant first completing the networks and buffer and then moving to the larger algorithm. My last step was visualizing both the MuJoCo environment's and my network's performance. Lastly, I chose to reuse a PER buffer as they are quite complex and not a core feature of my implementation paper (they were introduced by SAC). My experiments in order of priority were:

- Train SOP on the basic MuJoCo Environments, Hopper, Ant, Cheetah. This will serve to demonstrate the success of my SOP re-implementation.
- Train SOP+ERE on the basic MuJoCo Environments, Hopper, Ant, Cheetah. This will serve to demonstrate the success of my SOP re-implementation
- Analyze the effect of G Normalization on Environments. This will demonstrate my understanding of Normalization in the Continous RL space.
- Train SOP and SOP+ERE on the difficult Humanoid environment. This will demonstrate implementation success and likely successful hyperparameter tuning.
- (if time) Train on other Environments to demonstrate re-implementation success and hyperparameter search.
- (if time) Re-implement and Utilize the Random Perturbation Algorithm on Hopper. This will further demonstrate an understanding of Normalization in the RL Space.

8. Implementation details

The Original Authors of my paper published their code on Github. To this end, I reused their PER Buffer and a function that initializes network weights. Beyond this, the Q-networks, policy network, ERE Buffer, Logger and actual SOP training algorithm were all programmed myself. I used Gilbreth for GPU support as it allowed me to train multiple jobs simultaneously through slurm. Network calculations and training are done using the Pytorch engine. Finally, I used Gymnasium, a fork of the OpenAI gym library, to simulation all MuJoCo environments.

9. Results and Interpretation

9.1. SOP with and without G Normalization

In my first experiment I seek to repeat a modified version of the Normalization Experiment done in section 5.6. Instead of using SAC and Entropy Loss, I evaluate the SOP algorithm, with and without G Normalization. I choose to evaluate on the Humanoid-v4 Environment. As seen in Figures 11 and 12, G Normalization is critical in achieving exploration. As the conclusion of training, the network without G Normalization achieves 10% of the maximum reward achieved by the Normalized network. Notably these results prove that the Squashed Exploration Problem exists beyond the SAC algorithm.

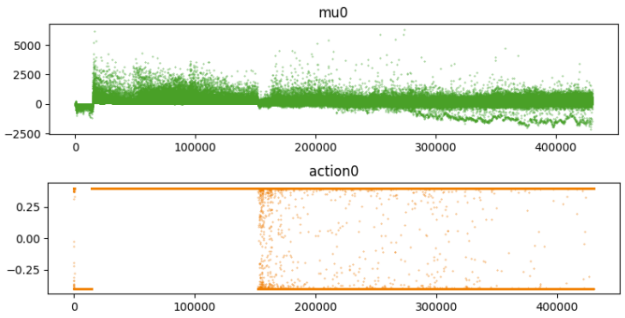


Figure 11. Top: The first output of the Policy Network over accross updates without G normalization. Note that the magnitudes are in the 1000s. Bottom: The corresponding actions taken by the policy network. There is almost no exploration

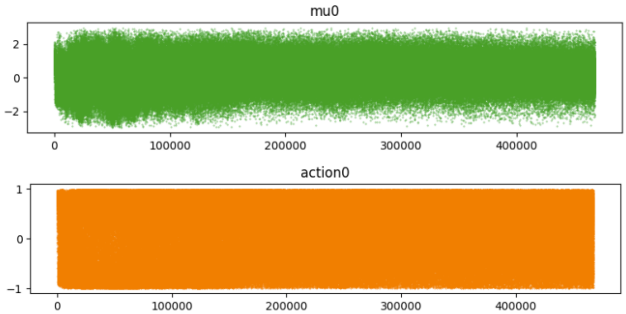


Figure 12. Top: The first output of the Policy Network over accross updates with G normalization. Note that they the magnitude is at most 2.3. Bottom: The corresponding actions taken by the policy network. Significant exploration occurs

Environment	My SOP+ERE Max	Original SOP+ERE
Hopper-v4	2700	3000
Cheetah-v4	8700	10000
Ant-v4	4000	4700
Humanoid-v4	5900	7000

Table 1. My Implementation of SOP ERE compared to the Original. Each value is the maximum episode reward accrued during the training period

9.2. My SOP's Performance

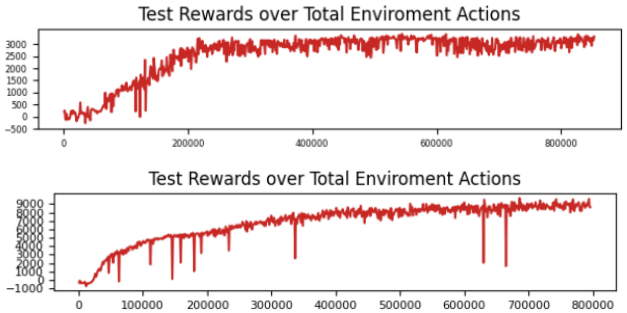


Figure 13. My SOP's Performance on Hopper (Top) and Cheetah (Bottom)

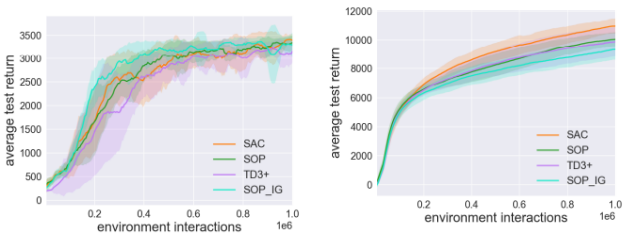


Figure 14. (Che et al., 2020)'s Performance on Hopper (Left) and Cheetah (Right)

In the following experiments I evaluate my model with a reused PER buffer. I seek to successfully train on two environments, Hopper-v4 and HalfCheetah-v4. The goal in each case is to teach the stick figure to move forward efficiently. Ultimately my stick figures are trained successfully and manage to achieve very results within 1000 points of the original paper (Figures 13, 14). During this experiment I noticed that my results were much noiser than the original paper. I suspect this is because I am using a different method to average my episodic rewards as the original method was not provided by (Che et al., 2020).

9.3. My SOP ERE's Performance

The following experiment compares the max rewards attained by my SOP+ERE algorithm to the original. I train my SOP+ERE algorithm on 4 environments in the original paper. Due to computational resources I was not always able to train for as many steps as the original authors. In such cases (Humanoid and Ant), I compare my results to the maximum achieved by the original paper in the same amount of steps. Table 1 shows that my model preforms 10.0%-15.8% worse than the original. I suspect this is because the original paper found specialized hyper-parameters for each Environment. In my case I was only able to perform a grid search of the Humanoid, which failed continuously until I lowered its learning rate to 8e-5.

9.4. Personal Environments

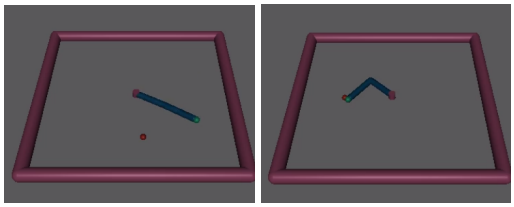


Figure 15. Left: The Starting state of the Reacher Environment, Right: My Trained Reacher immeadiately moving to the target (Usually within a second)

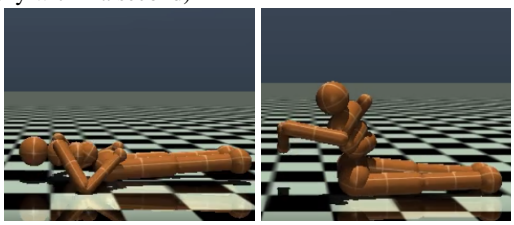


Figure 16. Left: The Starting state of HumanoidStandup, Right: The Local Optimum reached by my model. The policy learns to stand up, but refuses to do so forcefully

I was also able to train on two environments, Reacher-v4 and HumanoidStandup-v4, which were not trained in the original paper. Reacher-v4 seeks to move its arm towards a target that spawns randomly nearby. HumanoidStandup-v4 is the Humanoid-v4 Enviroment, but the agent starts off laying on it's back. I was able to achieve very satisfactory results on Reacher-v4 within 20,000 steps (see figure 15). However HumanoidStandup-v4 Proved much more difficult. Despite my best efforts I was only able to get it to sit up (see figure 16). I have two theories as to why the policy cannot escape this local optimum. Ironically, it is possible that regularization may prevent progress in this situation. In order to stand up, large movements would need to occur at multiple joints. This is actively prevented by the aforementioned G Normalization. Secondarily, it is possible that the agent is not sufficiently complex to learn coordinate the movement required to stand up since I only used 4 layers for π.

10. Discussion and Conclusion

10.1. Limitations

My greatest limit was training time. My algorithm requires 3 networks to train simultaneous on a reasonably complex physics simulation. Furthermore, in order to understand how the agent is behaving, continous rendering of said physics simulation is required. Given these constraints Training often took upwards of 6 hours, with the Humanoid taking 9 to fully converge.

10.2. Conclusion

The Streamlined Off Policy algorithm is an incredibly powerful method for solving continous RL tasks. In this paper I reiterate that it is capable of training on multiple complex physics simulations, achieving desirable results on almost all of them. Finally, I conduct an analysis that demonstrates that G Normalization is a remarkably simple method that achieves results similar to the much more complex Entropy Loss.

References

Ahmed, Z., Roux, N. L., Norouzi, M., and Schuurmans, D. Understanding the impact of entropy on policy optimization. *International Conference on Machine Learning*, pp. 151–160, 5 2019. URL http://proceedings.mlr.press/v97/ahmed19a/ahmed19a.pdf.

Chapelle, O. and Wu, M. Gradient descent optimization of smoothed information retrieval metrics. *Information Retrieval Journal*, 13(3):216–235, 9 2009. doi: 10.1007/s10791-009-9110-3. URL https://doi.org/10.1007/s10791-009-9110-3.

Che, W. and Ross, K. W. Boosting Soft Actor-Critic: Emphasizing Recent Experience without Forgetting the Past. *arXiv (Cornell University)*, 6 2019. doi: 10.48550/arxiv.1906.04009. URL http://arxiv.org/abs/1906.04009.

Che, W., Wu, Y., Vuong, Q., and Ross, K. W. Striving for simplicity and performance in off-policy DRL: Output Normalization and Non-Uniform Sampling. *ICML 2020*, 1:10070–10080, 7 2020.

Fujimoto, S., Van Hoof, H., and Meger, D. Addressing function approximation error in Actor-Critic methods. *ICML 2018*, 2 2018. URL http://export.arxiv.org/pdf/1802.09477.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *arXiv (Cornell University)*, 1 2018. URL https://arxiv.org/pdf/1801.01290.pdf.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv (Cornell University)*, 9 2015. URL http://export.arxiv.org/pdf/1509.02971.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing Atari with Deep Reinforcement Learning. *arXiv (Cornell University)*, 12 2013. URL http://cs.nyu.edu/~koray/publis/mnih-atari-2013.pdf.

Sun, H. Reinforcement Learning in the Era of LLMs: What is Essential? What is needed? An RL Perspective on RLHF, Prompting, and Beyond. *arXiv (Cornell University)*, 10 2023. doi: 10.48550/arxiv.2310.06147. URL https://arxiv.org/abs/2310.06147.

Watkins, C. Learning from delayed rewards. *Cambridge*, 1 1989. URL http://ci.nii.ac.jp/naid/10010359770/.