FINAL PROJECT REPORT: QUALITATIVE AND QUANTITATIVE ANALYSIS OF CONFIGURATING SAC ALGORITHM WITH REDQ AND RESET MECHANISM

Bale Chen
New York University Shanghai

ABSTRACT

This paper presents a study on the Soft-Actor Critics (SAC) algorithm, one of the most fundamental methods in deep reinforcement learning. The study probes into multiple algorithmic aspects and configurations of SAC and applies recent advancements such as Randomized Ensembled Double-Q Learning (REDQ) and reset mechanism to the classic SAC algorithm. The authors compare the performance of different variants of SAC trained for different epochs using both quantitative and qualitative analysis of their performance on Hopper-v3 and HalfCheetah-v3 environments by MuJoCo. The results show that the variant adopting REDQ and resetting algorithm outperforms all other variants in both environments with higher average test episode return. In ablation studies, we also found that REDQ is a generally more robust mechanism to improve sample efficiency while the contribution of reset mechanism is insignificant.This study also shows how configuration details would alter SAC's performance.

1 INTRODUCTION

In deep reinforcement learning (DRL), model-free algorithms have been given immense attention in recent years. They have achieved great performance in both discrete and continuous controls (Badia et al., 2020)(Smith et al., 2022). Soft-Actor Critics (SAC) (Haarnoja et al., 2018) algorithm is one of the most fundamental method upon which multiple other algorithms have been built. It takes advantage of double-Q learning, entropy-regularization, and experience replay to perform well in various domains.

In this project, we probe into multiple algorithmic aspects and configurations of SAC and applied the recent advancements such as Randomized Ensembled Double-Q Learning (REDQ) (Chen et al., 2021) and reset mechanism (Nikishin et al., 2022) in DRL to the classic SAC algorithm. Both REDQ and reset mechanism deals with the issue of high Update-to-Data (UTD) ratio, but from different perspectives. According to Chen et al. (2021), REDQ is able to reduce not only the average Q estimation bias, but also the standard deviation of the bias. Reset mechanism, differently, focuses on reducing the primacy bias that hinders the agent to learn from later stage data (Nikishin et al., 2022). Both of them allow a higher UTD ratio which leads to a higher sample efficiency. In this project, we also try to implement these two algorithms together on an SAC agent.

2 QUANTITATIVE AND QUALITATIVE ANALYSIS OF SAC VARIANTS

In this section, we compare three SAC models trained for 0, 100, 300 epochs using both quantitative analysis of their performance and qualitative analysis of the rendered images. The experiments are conducted on Hopper-v3 and HalfCheetha-v3 environments by MuJoCo. Besides, action noise in rendering stage is also experimented in the following experiment. A new variant of the model adopting REDQ and resetting algorithm is also proposed and compared in the analysis. In this variant, we use a higher UTD ratio of 9, 10 randomized ensembled Q-functions, and 3 resets over the whole 300-epoch training process.

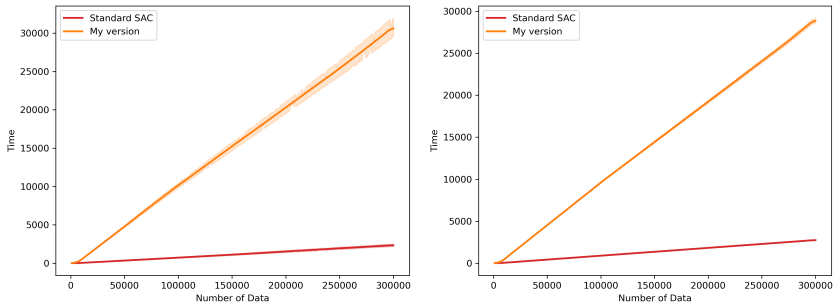
2.1 Quantitative Analysis

First, the average test episodes return is demonstrated in Table 1. We observe that SAC-REDQ-reset reports a significantly better performance than all other variants in both environments. In the Hopper-v3 environment, this improvement is rather outstanding since SAC-REDQ-reset have a over 3x higher average test episode return than the standard SAC trained for 300 epochs. In terms of training epochs, unsurprisingly SAC trained for 300 epochs performed better than one with 100 epochs, and the random agent performs the worst.

	SAC-0	SAC-100	SAC-300	SAC-REDQ-reset
Hopper-v3	47.80637	615.13385	1165.41	**3544.78**
HalfCheetah-v3	-438.56	5192.87	7318.76	**8312.383**

Table 1: The average test epsiodes return of four variants of SAC model. "SAC-n" denotes an SAC model trained for n epochs. "SAC-REDQ-reset" is the variant that is proposed in this project.

We then look into the training time for each variant. SAC-REDQ-reset used almost ten times more wall clock time than the standard SAC model. It generally takes around 8.4 hours to train a SAC-REDQ-reset agent and around 0.8 hours train a standard SAC agent for 300 epochs.



(a) Hopper-v3 (b) HalfCheetah-v3

Figure 1: Wall clock time

2.2 Qualitative Analysis

We first look at the random agent (i.e. the agent that has not been trained yet). The actions will be totally random since the network parameters of the agent is randomly initialized. The results are demonstrated in Figure 2. In Hopper-v3 environment, the hopper don't even know how to move forward, and quickly terminates because of an unhealthy angle in the joints. In the HalfCheetah-v3 environment, the agent put random pressure on the joints and the half cheetah ends up rolling around at the initial place without making any forward movement.

Then, we train the agent for 100 and 300 epochs with standard SAC algorithm. For both number of training epochs, the agent acquires the right motion to move forward in both environments. In HalfCheetah-v3, the motion done by an agent trained for 100 epochs and 300 epochs are mostly identical. But with respect to the environment steps, the agent trained for 300 epochs is running faster than the one trained for 100 epochs. Agent trained for 100 epochs and 300 epochs can run for 1000 environment steps without falling down. In Hopper-v3, the agent trained with 100 epochs fail to control its gravity center after one hop, thus falling down in the second hop. Yet, it learns the basic motion of hopping within 100 epochs. The agent trained with 300 epochs managed 3 hops but still falls down at the end. A sample successful movement in both environment is demonstrated in Figure 3.

In terms of action noise in rendering, we didn't observe a noticeable difference in the behavior pattern. However, as we compare the the number of steps taken for the agent to move out of the

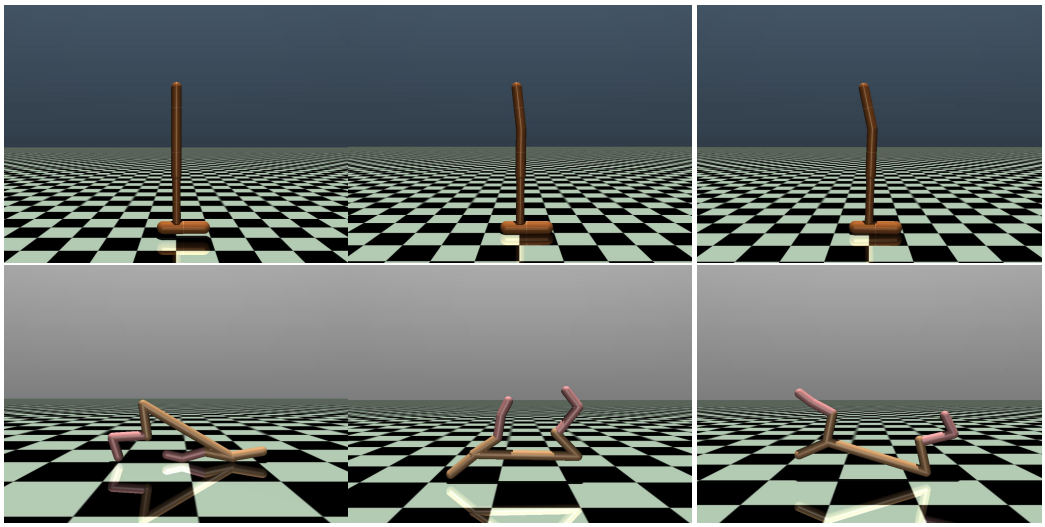


Figure 2: Rendered images of the random agent (Upper: Hopper-v3 Environment, Lower: HalfCheetah-v3 Environment)

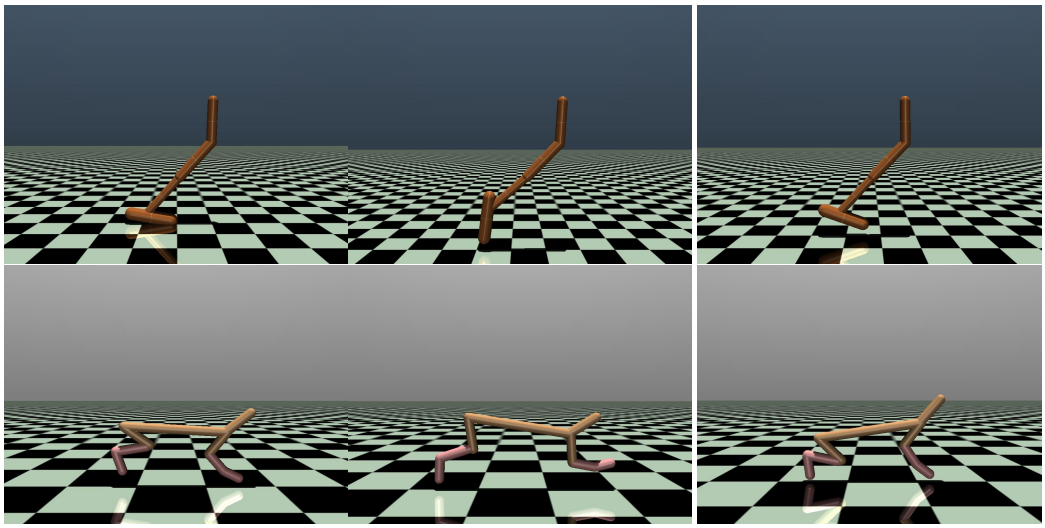
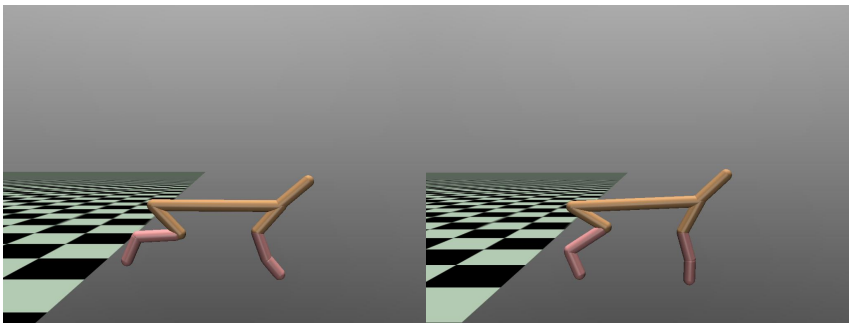


Figure 3: A successful forward movement in both environment using SAC trained with 100 epochs (Upper: Hopper-v3 Environment, Lower: HalfCheetah-v3 Environment)

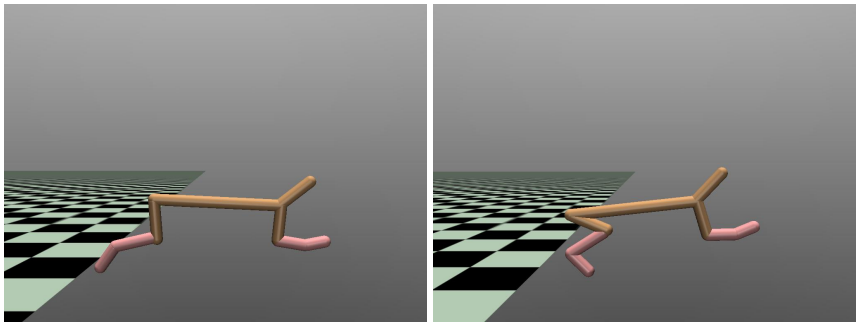
boundary in Figure 4, we found that with action noise, the agent is moving forward slower than the one without action noise.

Lastly, we look at our proposed agent SAC-REDQ-reset's renderings. We found that in both environment, the agent reached the maximal number of environment steps in the rendered episode. The agent successfully learnt how to hop without falling down in the Hopper-v3 environment, which echos the significant improvement of average test episode return mentioned in the quantitative comparison. In the HalfCheetah-v3 environment, no obvious disparity in behavior pattern is found. We then compare the number of steps taken for the agent to move out of the boundary in Figure 4. Our proposed agent takes less step to move outside of the platform boundary, which indicates that it's moving faster than the other variants.

Through an quantitative and qualitative analysis, we found that the proposed SAC-REDQ-reset performed significantly better than the standard SAC model in both environments and in multiple as-



(a) SAC-100, Environment Step #164 (b) SAC-300, Environment Step #135



(c) SAC-300-action-noise, Environment Step #145 (d) SAC-REDQ-reset, Environment Step #128

Figure 4: Rendered images of the random agent (Upper: Hopper-v3 Environment, Lower: HalfCheetah-v3 Environment)

pects. As changing so many components of SAC is hard to dissect where the improvement actually comes from, the subsection 3.4 and 3.5 in the ablation study further analyze this configuration.

3 Ablation Study of SAC

In this section, multiple configurations are experimented to examine how learning rate (α), discount rate (γ), polyak update rate (polyak), Update-To-Data (UTD) rate, and Halfway Reset (HR) would affect the SAC algorithm. The metrics used are performance (average test episodes return), average Q-value, and average normalized Q bias.

The default value for each hyperparameter of interest are as follow:

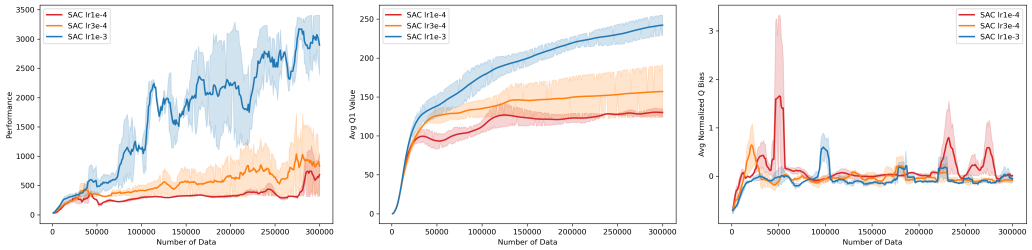
$$\alpha = 3 \times 10^{-4}\ ,\ \gamma = 0.99\ ,\ \ \text{polyak} = 0.995\ ,\ \ \text{UTD} = 1\ ,\ \ \text{HR} = \text{False}$$

3.1 Learning Rate α

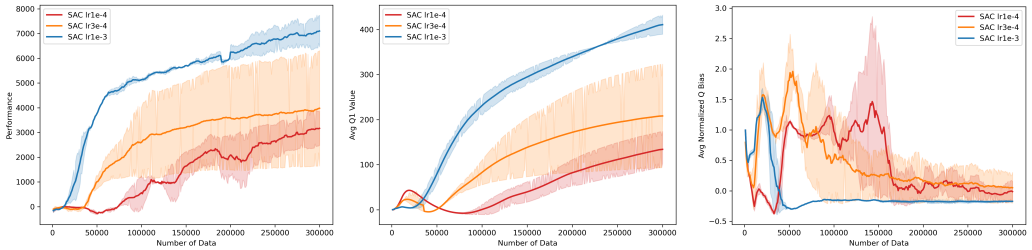
First, we compare the learning rate of $10^{-4}, 3 \times 10^{-4}, 10^{-3}$, holding other hyperparameters default. The results are shown in Figure 5.

From Figure 5(a) and 5(d), $\alpha = 10^{-3}$ gives a significantly better result than the other two, while the default version performs slightly better than $\alpha = 10^{-4}$. This order coincides with the order of Average Q-value in Figure 5(b) and 5(e), showing a positive correlation between model performance and average Q-value. This result indicates that a lower learning rate could cause the model to converge to a local maximum of Q-value. In contrast, using a higher learning rate like $\alpha = 10^{-3}$ could find a higher Q-value, resulting in a better policy and performance.

In terms of normalized Q bias in Figure 5(c) and 5(f), we observe that in Hopper environment, there's no obvious disparity among three curves, though $\alpha = 10^{-4}$ manifests a slightly larger variance. These three curves all converge approximately to 0 at the end. In the HalfCheetah environment,



(a) Performance, Hopper (b) Average Q-value, Hopper (c) Normalized Q Bias, Hopper



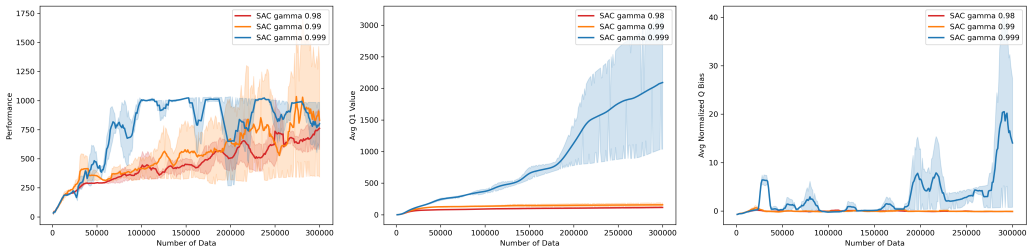
(d) Performance, HalfCheetah (e) Average Q-value, HalfCheetah (f) Normalized Q Bias, HalfCheetah

Figure 5: Comparing learning rate of 10^{-4}, 3×10^{-4}, and 10^{-3}.

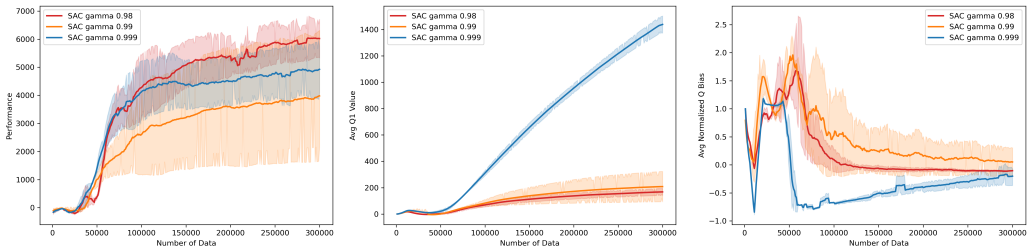
$\alpha = 10^{-3}$ displays a faster and stabler convergence of Normalized Q bias. However, unlike the other two configurations, $\alpha = 10^{-3}$ also makes the model underestimates the Q-value.

3.2 Discount Rate γ

Next, discount rate is also an important hyperparameter in reinforcement learning. We used $\gamma = 0.98,\ 0.99,$ and 0.999 in this experiment. A higher discount rate means a higher importance of the short-term rewards, while a lower discount rate values rewards in the future more.



(a) Performance, Hopper (b) Average Q-value, Hopper (c) Normalized Q Bias, Hopper



(d) Performance, HalfCheetah (e) Average Q-value, HalfCheetah (f) Normalized Q Bias, HalfCheetah

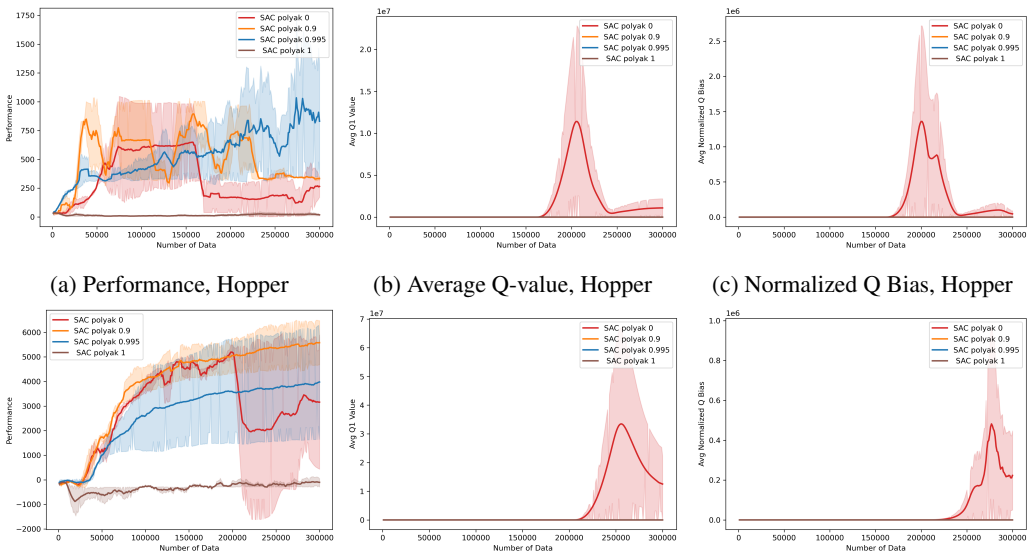
Figure 6: Comparing Discount Rate of 0.98, 0.99, 0.999.

From Figure 6, we see the two environments generates rather different results. In Hopper environments, the three configurations eventually offer a similar result in performance. Using $\gamma = 0.999$ gives a much better performance at the beginning but quickly reaches a bottleneck and becomes unstable. $\gamma = 0.99$ has a increasing variance in performance, while $\gamma = 0.98$ has a much steady growth. Whereas in HalfCheetah environment, the performance ranking is much clearer that $\gamma = 0.98$ is the best and $\gamma = 0.999$ is the worst. Regarding Q-value, it's positively correlated with γ value with $\gamma = 0.999$ growing much faster than the other two. In Hopper environment, the model with $\gamma = 0.999$ is evidently overestimating the Q-value in terms of the normalized Q bias, while in HalfCheetah environment, the model is underestimating.

This observation is likely to stem from the different environment settings. Possibly, in Hopper environment, the agent will not estimate the action-value accurately if it puts too much emphasis on short-term rewards. In Cheetah environment, having a short-term or long-term vision wouldn't affect too much on the estimation of Q-value. Generally, $\gamma = 0.98$ is preferred as it gives a fairly well and stable performance.

3.3 Polyak Update Rate

In this experiment, we investigate the influence of polyak update rate on the learning process. The values chosen are $0, 0.9, 0.995$, and 1. Using $\text{polyak} = 0$ means the target Q network would be the same as the Q-function that we train. $\text{polyak} = 1$ means the target Q network is fixed from the initialization. Results are shown in Figure 7 Seeing from Figure 7, $\text{polyak} = 0.995$ performs the



(a) Performance, Hopper (b) Average Q-value, Hopper (c) Normalized Q Bias, Hopper

(d) Performance, HalfCheetah (e) Average Q-value, HalfCheetah (f) Normalized Q Bias, HalfCheetah

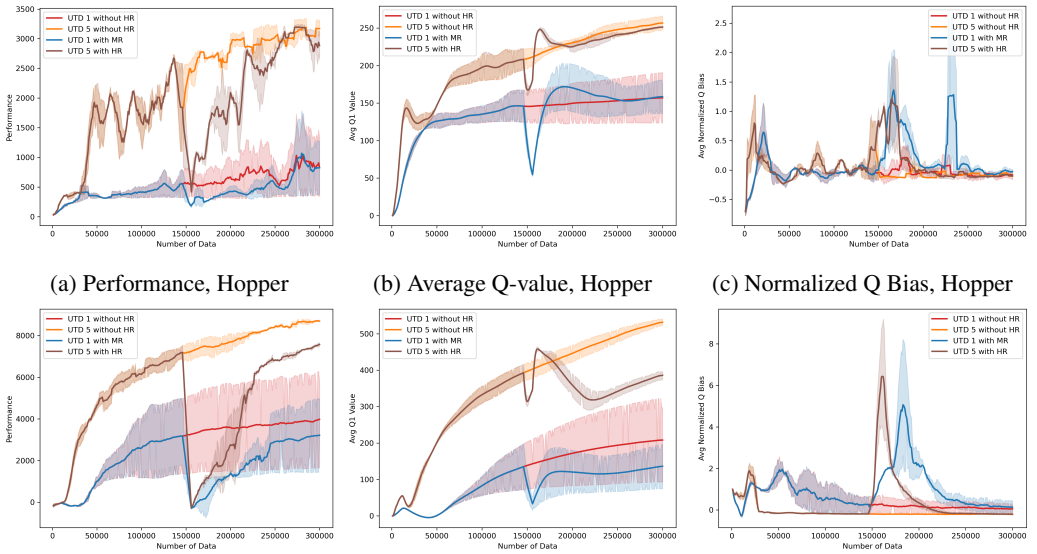
Figure 7: Comparing Polyak update rate of 0, 0.9, 0.995, and 1.

best in Hopper environment, yet $\text{polyak} = 0.9$ performs the best in HalfCheetah environment. In both environment, $\text{polyak} = 0$ performs as good in the first half of the process, but then suffers from a huge downgrade. This is justified by the average Q-value and normalized Q bias graph where there is a sudden overestimation of the Q-value around 150-200 epochs when $\text{polyak} = 0$. $\text{polyak} = 1$ as expected has the worst performance since the training target is completely arbitrary.

The different ranking of $\text{polyak} = 0.9$ and $\text{polyak} = 0.995$ is a likely result from the difference in environments. $\text{polyak} = 0$ falls short because it will render a high dependence between target Q network and the Q network that we are training since they share the same set of parameters. The minimization process of loss function will thus become unstable and result in the sudden aforementioned breakdown.

3.4 Update-To-Data Ratio & Halfway Reset

In this experiment, UTD ratio and halfway reset trick are experimented together. We have for experiment groups: UTD $= 1$ and HR $=$ False, UTD $= 5$ and HR $=$ False, UTD $= 1$ and HR $=$ True, UTD $= 5$ and HR $=$ True.



(a) Performance, Hopper (b) Average Q-value, Hopper (c) Normalized Q Bias, Hopper

(d) Performance, HalfCheetah (e) Average Q-value, HalfCheetah (f) Normalized Q Bias, HalfCheetah

Figure 8: Comparing UTD Ratio of 1 and 5 with or without HR.

According to Figure 9, we observe that UTD $= 5$ performs much better than UTD $= 1$, while using or not using halfway reset doesn't make a different. No improvement were observed from any of the metrics after using the halfway reset trick. Even, in the case of HalfCheetah, after reset, the performance is degraded. The Q-value seems to be stuck in a local minimum after reset.

Higher UTD guarantees a higher sample efficiency and a higher frequency of updates. The Q function finds a better maximum point much faster and thus generates a better policy network, which explains its significantly better performance.

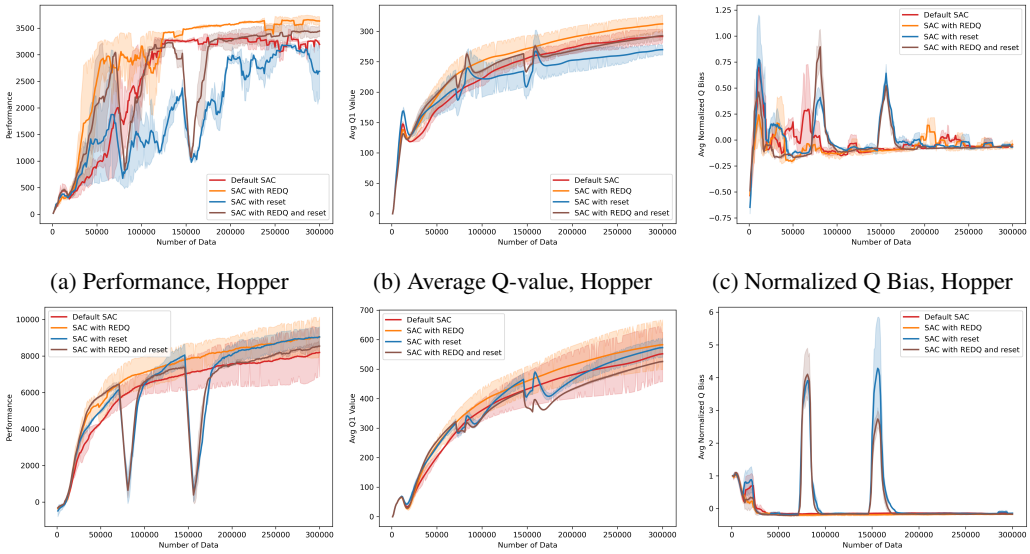
3.5 REDQ and Reset Mechanism

In this subsection, we further look into the success of SAC-REDQ-reset variant to address the question of where the improvement actually come from and whether REDQ coordinates well with resetting. Here we implement the full version of the reset mechanism proposed by Nikishin et al. (2022) instead of a simplified halfway reset.

We see that in both environment SAC with REDQ performed the best, though SAC with reset caught up in the HalfCheetah-v3 environment. The SAC with REDQ and reset performed unexpectedly mediocre. Especially, in HalfCheetah-v3, SAC with REDQ and resetting has a performance score lower than both SAC with only REDQ and SAC with only resetting. This result indicates that the success of SAC-REDQ-reset mostly comes from REDQ algorithm and the combination of both algorithm doesn't bring about any benefit if not negative impact.

4 Conclusion

In this project, we probed into how some configuration details wound affect SAC's performance on Hopper-v3 and HalfCheetah environment. Also, a combination of the two recent advancement in DRL algorithms, REDQ and resetting, are combined and experimented together. We found that REDQ is generally a more robust algorithm that allows a higher UTD ratio and thus higher sample



(a) Performance, Hopper (b) Average Q-value, Hopper (c) Normalized Q Bias, Hopper

(d) Performance, HalfCheetah (e) Average Q-value, HalfCheetah (f) Normalized Q Bias, HalfCheetah

Figure 9: Comparing UTD Ratio of 1 and 5 with or without HR.

efficiency. Resetting mechanism doesn't contribute as much in these two environments. The existence of primacy bias in these two environment and our learning settings awaits further investigation.

In terms of other hyperparameters, generally, a higher learning rate $\alpha = 10^{-3}$ and higher UTD ratio leads to better performance in different environment settings. As for other hyperparameters, they are dependent on the environment so that fine-tuning for different environment is necessary in actual implementation.

REFERENCES

Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Zhaohan Daniel Guo, and Charles Blundell. Agent57: Outperforming the atari human benchmark. *CoRR*, abs/2003.13350, 2020. URL https://arxiv.org/abs/2003.13350.

Xinyue Chen, Che Wang, Zijian Zhou, and Keith W. Ross. Randomized ensembled double q-learning: Learning fast without a model. *CoRR*, abs/2101.05982, 2021. URL https://arxiv.org/abs/2101.05982.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018. URL http://arxiv.org/abs/1801.01290.

Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning, 2022. URL https://arxiv.org/abs/2205.07802.

Laura Smith, Ilya Kostrikov, and Sergey Levine. A walk in the park: Learning to walk in 20 minutes with model-free reinforcement learning. *ArXiv*, abs/2208.07860, 2022.