Understanding Off-Policy Control

Xinhao Liu

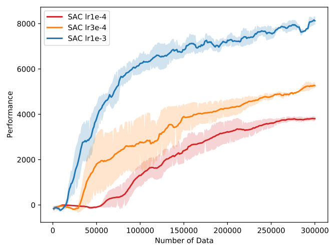
New York University Shanghai

1 Introduction

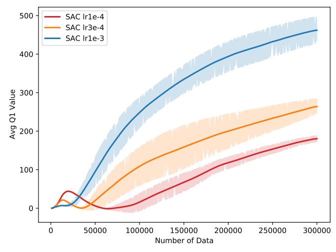
In this report, we analyze and understand state-of-the-art deep reinforcement learning algorithms in off-policy control by experiments. In Sec. 2, most of the analysis and discussion is based on the SAC Haarnoja et al. (2018). We discuss how different hyperparameters affect the performance of the algorithm. In Sec. 3, we propose a variant of SAC with the inspiration from RED-Q Chen et al. (2021). We also render the agent and compare the training results at different epochs. The code of this project can be found at: https://github.com/Gaaaavin/REDQ-fall22-student

2 Hyperparameters

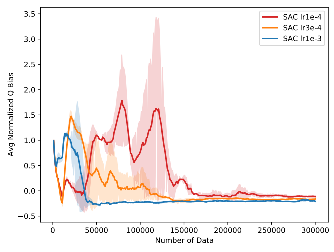
2.1 Learning Rate



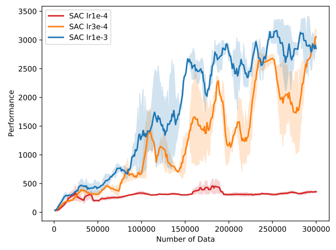
(a) Performance, Half Cheetah



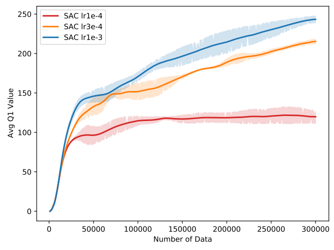
(b) Q value, Half Cheetah



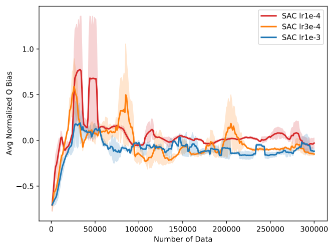
(c) Q bias, Half Cheetah



(d) Performance, Hopper



(e) Q value, Hopper



(f) Q bias, Hopper

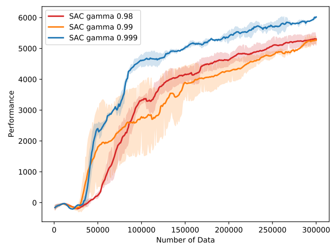
Figure 1: **Various learning rate**. The figures report performance, Q value, and Q bias of SAC when trained with different learning rates. Experiments are run in two environments.

We discuss the influence of different learning rate on the SAC Haarnoja et al. (2018) algorithm. The default learning rate is $\alpha = 3 \times 10^{-4}$. In the experiment, we also use $\alpha = 1 \times 10^{-3}$ and $\alpha = 1 \times 10^{-4}$ for comparison. In general, as shown in Fig. 1, the learning rate $\alpha = 1 \times 10^{-3}$ has the best performance, whereas $\alpha = 1 \times 10^{-4}$ does not perform very well. One possible reason is that SAC has a low update-to-data (UTD) ratio ($G = 1$). Hence, when training with a low learning rate, the performance converges very slowly. Another explanation that the result is due to the nature of stochastic gradient descent, where the training tend to stuck at a local minimum when using a small learning rate. This is most likely for the Hopper environment because the performance does not increase with the growing number of data when $\alpha = 1 \times 10^{-4}$.

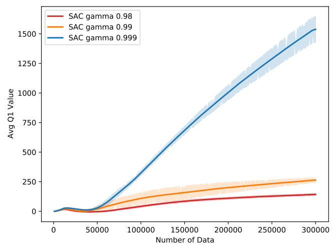
Comparing the performance and Q value curve, we can find a clear correlation that a higher Q value results a better performance. It is interesting to observe that in the Half Cheetah environment, the SAC algorithm has an overestimation for the Q values with all the learning rates, as shown in Fig. 1b. Still, the highest learning rate ($\alpha = 1 \times 10^{-3}$) has the minimal overestimation and corrects it very quickly in the training progress.

The two environment varies a lot in terms of the averaged normalized Q bias. In Half Cheetah, $\alpha = 1 \times 10^{-3}$ is the fastest to reach a very low bias. This also agrees with the Q value and the performance, though all learning rates have a slight underestimation when converged. On the other hand, there is no significant difference in the Q bias among the three learning rates in the Hopper environment. It is worth noting, however, that all learning rates have an underestimation of the Q values at the beginning of the training. This might indicate that Hopper is a simpler environment then Half Cheetah, which is proved to be true after checking the documentation of the Gym library.

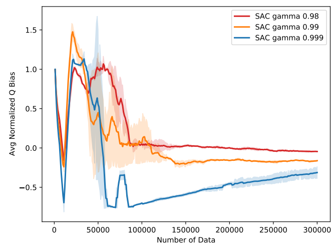
2.2 Discount Rate



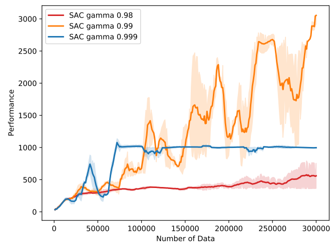
(a) Performance, Half Cheetah



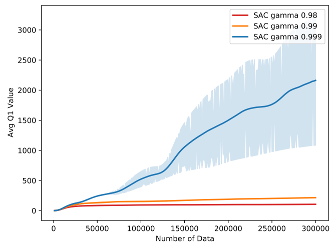
(b) Q value, Half Cheetah



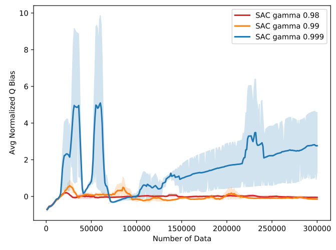
(c) Q bias, Half Cheetah



(d) Performance, Hopper



(e) Q value, Hopper



(f) Q bias, Hopper

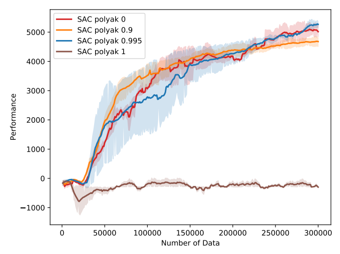
Figure 2: **Various discount rates**. The figures report performance, Q value, and Q bias of SAC when trained with different discount rates. Experiments are run in two environments.

In this section, we discuss with different discount rates when using the same learning rate. The default discount rate is $\gamma = 0.99$. We compare it with $\gamma = 0.98$ and $\gamma = 0.999$, and the result is shown in Fig. 2. We found that the results are different in the two environments. In Half Cheetah, $\gamma = 0.999$ has the best performance, and $\gamma = 0.99$ has the worst performance. On the other hand, in Hopper, $\gamma = 0.99$ has the best performance and $\gamma = 0.98$ has the worst performance. Moreover, both $\gamma = 0.98$ and $\gamma = 0.999$ saturate at the very beginning of the training. This could be explained by the complexity of the two environments. Since Hopper is simpler, the agent maybe can run beyond 1000 time steps before falling. Thus, paying more attention to later time steps does not lead to an improvement in the performance. In the opposite, it might hurt the training at the current learning rate. Nevertheless, in the more complex environment Half Cheetah, the agent may often falls before 1000 time steps. A larger discount rate may guide the algorithm to learn a policy that can run for longer time steps before falling, which significantly improves the performance (episodic return).

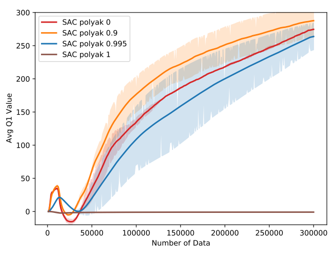
It is meaningless to discuss the Q value because different discount rate changes the definition of Q value. Thus, it is expected that the Q value with different discount rate is not comparable as shown in the figure.

The value of Q bias in the experiments agrees our hypothesis about the environment. In Half Cheetah, as shown in Fig. 2c, a larger discount rate ($\gamma = 0.999$) induces a significant underestimation of the Q value at the beginning, because the agent cannot run before falling for 1000 time steps. As the number of data increases, the algorithm learns a better policy so that the negative bias increases towards zero. In the opposite, in Hopper, as shown in Fig. 2f, a large discount rate lead to an increasing overestimation as the number of data grows. This is also the reason behind the saturation in the performance curve in Fig. 2c.

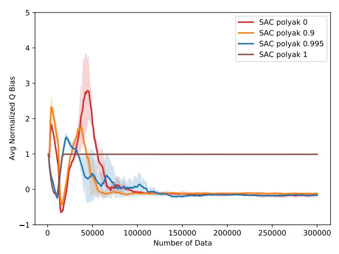
2.3 Polyak



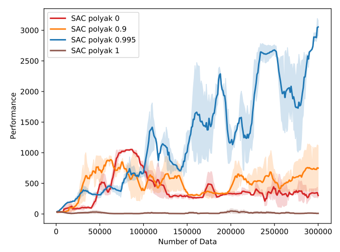
(a) Performance, Half Cheetah



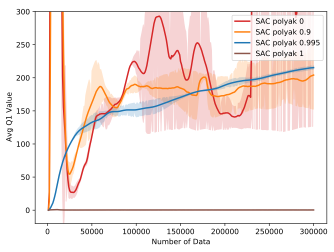
(b) Q value, Half Cheetah



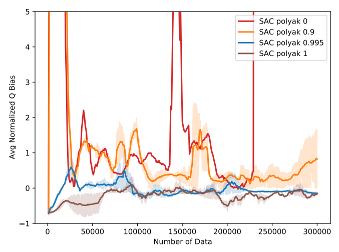
(c) Q bias, Half Cheetah



(d) Performance, Hopper



(e) Q value, Hopper



(f) Q bias, Hopper

Figure 3: **Various polyak values**. The figures report performance, Q value, and Q bias of SAC when trained with different polyak values. Experiments are run in two environments.

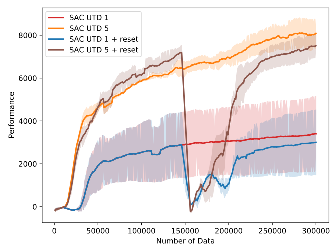
Using a target network to slow the update when computing the loss in order to stabilize the training is useful trick that is used in Lillicrap et al. (2015); Fujimoto et al. (2018); Haarnoja et al. (2018); Chen et al. (2021). In our code implementation, a high polyak value means a slower update of the target network. In the extreme cases, polyak value $p = 0$ means a synchronization of the target network and the source network, whereas $p = 1$ means never updating the target network. In the following discussion, we ignore $p = 0$ because this is theoretically incorrect and has the worst performance.

As shown in Fig. 3, in Half Cheetah, all the polyak value has a relatively similar final performance except $p = 0$. In Hopper, $p = 0.995$ leads to the best performance, and lower polyak value (0.9 and 0) does not perform very well.

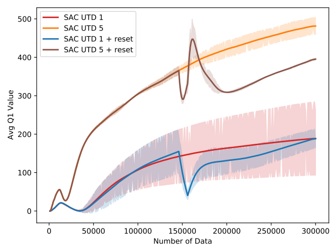
In terms of the average Q value, $p = 0.9$ leads to the highest Q value in Half Cheetah and $p = 0.995$ leads to a highest converged Q value. Observing the Q value and Q bias curve, we can find the lack of a target network ($p = 0$) makes the learning very unstable, especially in the case of Hopper.

2.4 UTD and Reset

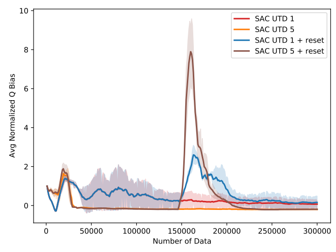
A significant improvement of Chen et al. (2021) over Haarnoja et al. (2018) is the increased UTD ratio ($G \gg 1$). Here, we compare $G = 1$ and $G = 5$ to get an idea of the effect of UTD radio. As shown in Fig. 4, $G = 5$ has a faster learning efficiency and a higher performance than $G = 1$. We combine this observation with the discussion in Chen et al. (2021) to see that SAC has a potential to learning with a UTD ration slightly higher than 1, while is not guaranteed to have good performance when $G \gg 1$. The faster learning efficient when $G = 5$ can also be proved from the Q value and Q bias curve. Although a higher UTD ratio leads to a larger overestimation at the beginning of the training, the bias can be corrected faster than with a lower UTD ratio.



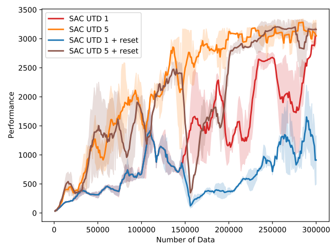
(a) Performance, Half Cheetah



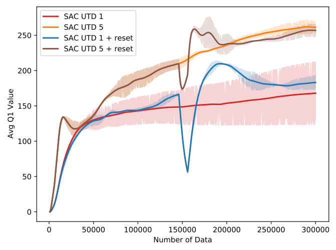
(b) Q value, Half Cheetah



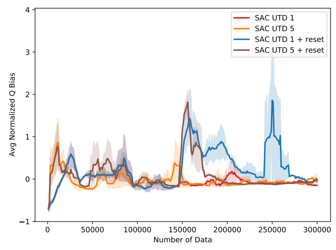
(c) Q bias, Half Cheetah



(d) Performance, Hopper



(e) Q value, Hopper



(f) Q bias, Hopper

Figure 4: **Various UTD and reset**. The figures report performance, Q value, and Q bias of SAC when trained with different UTD ratios and reset conditions. Experiments are run in two environments.

We also try to reset the networks at the middle of the training, as suggested by Nikishin et al. (2022). It turns out that it takes about the same number of data to regain the performance before the resetting. In Half Cheetah, the performance is slightly lower at the end of training compared to non-resetting. It might need more data for the resetting ones to outperform the original ones. In Hopper, however, we observe a significant loss of performance for resetting when $G = 1$. This can be explained by the low learning efficiency. Hence, it take much more number of data to regain the same performance as before the resetting.

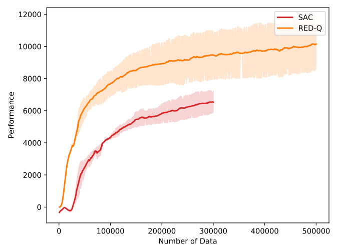
One significant observation from Fig. 4c and 4f is that the overestimation after the resetting is larger than that at the beginning. This can be explained by the high return data in the reply buffer at the middle of the training. Because the return in the data tend to be higher than those in the beginning of the training, the overestimation can be large and cost more data to correct. It implies that it might not be a good idea to incorporate resetting into high UTD ratio training. Nevertheless, as shown in Fig. 4d, a low UTD ratio also leads to the slow regaining of the performance. This contradiction is a good topic for further analysis and discussion with more experiments. Maybe we could try resetting for RED-Q and compare the performance with the results here.

3 Improvements and Comparisons

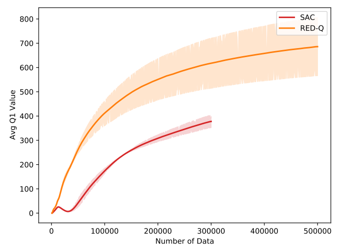
In this section, all the discussion is based on SAC Haarnoja et al. (2018) and a proposed variant of it. The details of the two algorithms are listed in Tab. 1. Also, to discover the potential of the proposed algorithm with more data, we train it 500 more epochs, compared to 300 epochs for SAC.

Table 1: **Hyperparameters.** The two algorithms discussed in this section has different hyperparameters as listed in the table. The proposed method will be referred to as RED-Q in the following texts.

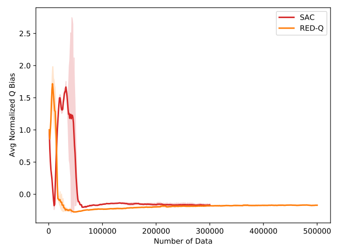
Algorithm	Learning rate	Batch size	Hidden size	Q functions	UTD ratio
SAC	3×10^{-4}	128	128	2	1
Proposed (RED-Q)	1×10^{-3}	256	256	4	5



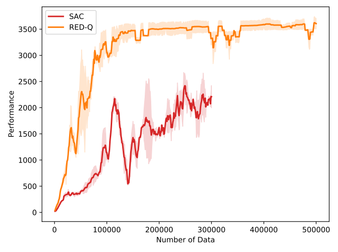
(a) Performance, Half Cheetah



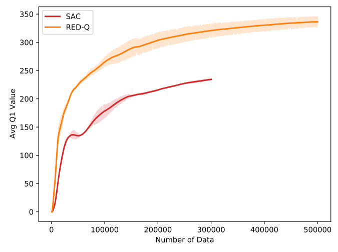
(b) Q value, Half Cheetah



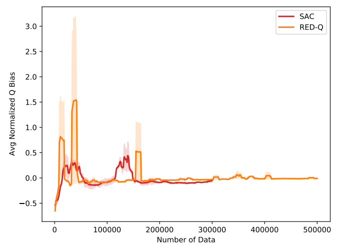
(c) Q bias, Half Cheetah



(d) Performance, Hopper

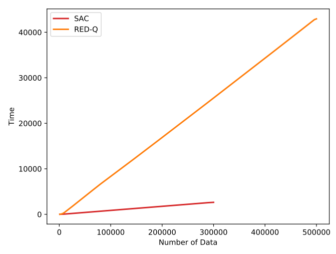


(e) Q value, Hopper

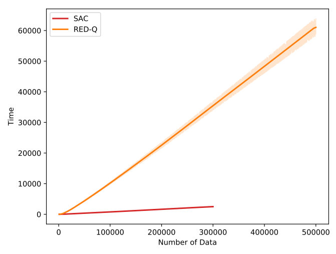


(f) Q bias, Hopper

Figure 5: **Experiment performance**. The figures report performance, Q value, and Q bias of SAC and the proposed method (RED-Q). Experiments are run in two environments.



(a) Time, Half Cheetah



(b) Time, Hopper

Figure 6: **Training time**. The figures report training time of SAC and the proposed method (RED-Q). Experiments are run in two environments.

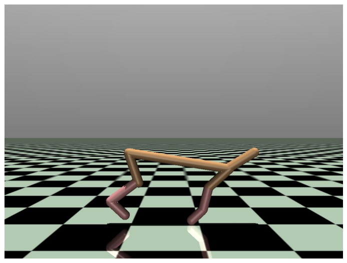
3.1 Performance

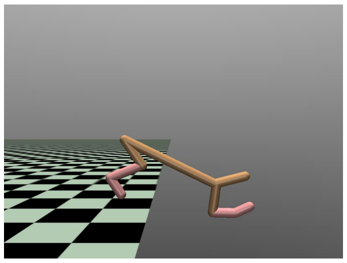
As shown in Fig. 5, the proposed method significantly outperforms SAC in all regards. In terms of performance, the proposed method can achieve a much higher average return when trained with the same number of data. This is mainly due to the high UTD ratio used in the training, as discussed in Sec. 2.4. Moreover, the proposed method has more Q functions than SAC, which avoids an overestimation at the beginning of the training. This can be witnessed both from the Q value curve and Q bias curve.

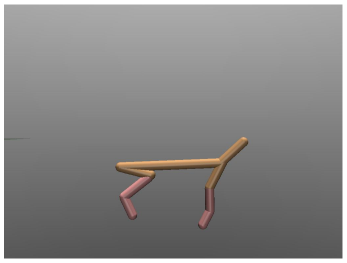
3.2 Efficiency

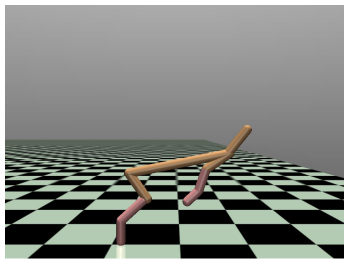
Fig. 5 also shows that increasing the number of data could also improve the performance, although marginally. It can also be seen from 5d that the improvement is negligible for RED-Q because it converges very efficiently with only about 100,000 number of data. We can infer that the improvements from more data is more beneficial to SAC than to RED-Q. However, it is worth noting that the sample efficiency of RED-Q is at the cost of computation efficient, as shown in Fig. 6. Because of larger UTD ratio and more Q-functions, RED-Q takes more computation time to train per data. In other words, with enough amount of data, SAC might achieve the same performance as the proposed method, and the computation time it takes for the two methods to achieve the same performance is roughly comparable. Still, we prefer RED-Q in real applications because sample efficiency is more important considering the wearing of robots.

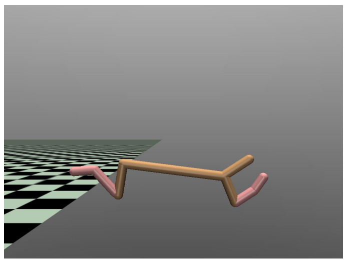
3.3 Qualitative Analysis


(a) SAC, epoch0, frame 180


(b) SAC, epoch100, frame 180


(c) SAC, epoch300, frame 180


(d) RED-Q, epoch100, frame 120


(e) RED-Q, epoch300, frame 120

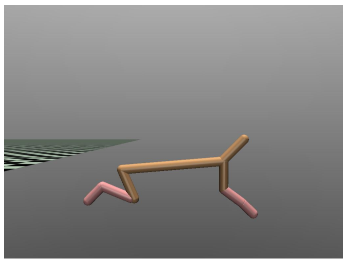

(f) RED-Q, epoch500, frame 120

Figure 7: **Agent rendering**. The figures show status of the agent at the same frame when trained with different algorithms and after different epochs. Entire animation can be found at Google drive.

In this subsection, we render the agent at certain epochs in the training progress. Fig. 7 shows the rendered agents. The specific frames are chosen for better illustration and comparison. Comparing the images in the first row, the agent moves a longer distance after more training epochs. The same is for RED-Q. Please refer to the complete animation in the Google drive to find that the agent trained with RED-Q moves faster after same epochs of training.

4 Conclusion

In summary, we can observe that the tuning of hyperparameters has some effects on the performance of the algorithms, the difference is marginal, especially for complex environment like Half Cheetah. We can infer that the difference is even negligible in more complex environments, as long as the hyperparameters are in a reasonable range. It is still useful, though, to understand and explain the failure mode of different algorithms and inspire the design of improved ones.

Moreover, we can find that RED-Q can perform a data-efficient training of control policy, but with larger computation requirements than SAC. Variants of RED-Q Hiraoka et al. (2021) is proposed to reduce the computation by introducing another form of regularization technique. Both algorithms are proved to work in controlling real robots, as discussed in Smith et al. (2022). In the future, it would be interesting to do more experiments on the pros and cons of the two algorithms.

References

Xinyue Chen, Che Wang, Zijian Zhou, and Keith Ross. Randomized ensembled double q-learning: Learning fast without a model. *arXiv preprint arXiv:2101.05982*, 2021.

Scott Fujimoto, Herke van Hoof, and Dave Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

Takuya Hiraoka, Takahisa Imagawa, Taisei Hashimoto, Takashi Onishi, and Yoshimasa Tsuruoka. Dropout q-functions for doubly efficient reinforcement learning. *arXiv preprint arXiv:2110.02034*, 2021.

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. *arXiv preprint arXiv:1509.02971*, 2015.

Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning. In *International Conference on Machine Learning*, pp. 16828–16847. PMLR, 2022.

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