INVESTIGATING HOW HYPERPARAMETERS INFLUENCE SAC PERFORMANCE

Yuxuan Wang

New York University Shanghai

ABSTRACT

Adequate hyperparameter tuning is critical to enhancing the performance of deep reinforcement learning models. Through carefully designed ablation study experiments, we implement an SAC-equivalent REDQ algorithm and test the influence of six hyperparameters in two MuJoCo environments of Ant and Hopper. The two environments reacts differently to the hyperparameters, and hyperparametering tuning should be specified with respect to the environment to achieve best agent performance. More challenging environments like Ant demands more complex algorithm setting for better optimization. α, polyak and batch size are relatively more influential in both environments. We verify that the agent performance agrees with the Q estimate produced by the target networks, and the distribution of Q estimate and entropy value is reasonable across all the states.

1 INTRODUCTION

The performance of deep reinforcement learning algorithms can be heavily influenced by the choice of hyperparameters. In this project, large-scale experiments are conducted to gain insights of how six different hyperparameters affect the behavior and performance of SAC, an off-policy control algorithm that is arguably the current most popular model-free off-policy baseline. We examine and assess the agent's performance with respect to these three criteria: agent's performance as discounted sum of all rewards, averaged normalized Q value bias and the standard deviation of that bias, and the last two criteria is introduced in Chen et al. (2021). In addition, we perform visualization of the Q estimates and entropy as defined in Haarnoja et al. (2018) and examine their relationship with agent performance.

2 RELATED WORK

In Chen et al. (2021), a model-free algorithm with a high sample efficiency called REDQ is implemented. The algorithm comes with a high Update-To-Data (UTD) ratio, which is the number of updates taken by the agent compared to the number of actual interactions with the environment. In addition, the algorithm involves multiple Q networks to perform Q value estimate and to compute the Q bias and std of the bias for agent performance evaluation. By turning the hyperparameters of REDQ and setting utd ratio = 1 and num Q = 2, we can get an algorithm that's equivalent to SAC. SAC stands for soft-actor-critic. It is an algorithm that optimizes a stochastic policy in an off-policy way and the approaches it takes are similar to the ones implemented in DDPG developed by Lillicrap et al. (2019). It also incorporates the clipped double-Q trick which was introduced in the TD3 algorithm developed by Fujimoto et al. (2018). Further illustration on SAC is provided in Section3.

3 METHOD

Designed by Haarnoja et al. (2018), SAC is a algorithm that optimizes two Q networks that estimates the value of state-action pairs, and a policy network that outputs the best action given the current state. It includes an entropy term to strike a balance between exploration and exploitation across all states, which is an critical feature of SAC. To stop the target network from shifting all the time during training, a polyak ratio is applied for updating the networks by keeping a portion of the old network setting. It performs MSBE minimization for Q functions so that they are more precise, and it performs gradient ascent on policy network for return maximization. The following pseudo code is given in Haarnoja et al. (2018).

Algorithm 1 Soft Actor-Critic

```
Initialize parameter vectors $\psi$, $\bar{\psi}$, $\theta$, $\phi$.
for each iteration do
    for each environment step do
        $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$
        $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
        $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$
    end for
    for each gradient step do
        $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
        $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$
        $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
        $\bar{\psi} \leftarrow \tau\psi + (1 - \tau)\bar{\psi}$
    end for
end for
```

Figure 1: pseudo code of SAC algorithm

We follow the same definition of the average normalized Q value bias and the standard deviation of the bias as in Chen et al. (2021). Let $Q^\pi(s, a)$ be the action-value function for policy π using the standard infinite-horizon discounted return definition and $Q_\phi(s, a)$ as an estimate of $Q^\pi(s, a)$, $i = 1, \ldots, N$. Then the bias of an estimate at state-action pair (s, a) to be $Q_\phi(s, a) - Q^\pi(s, a)$. When the analysis episodes are run with the current policy π, for each state-action pair visited, we obtain both the discounted Monte Carlo return and the estimated Q value using Q_ϕ, and then compute the difference to obtain an estimate of the bias for that state-action pair. The average and std of these bias values are then calculated. In addition, to deal with the significant changes of MC returns, we use the normalized bias of the estimate $Q_\phi(s, a)$ to be $(Q_\phi(s, a) - Q^\pi(s, a))/|E_{\bar{s},\bar{a}\sim\pi}[Q^\pi(\bar{s}, \bar{a})]|$, which is the bias divided by the absolute value of the expected discounted MC return for state-action pairs sampled from the current policy.

4 RESULTS

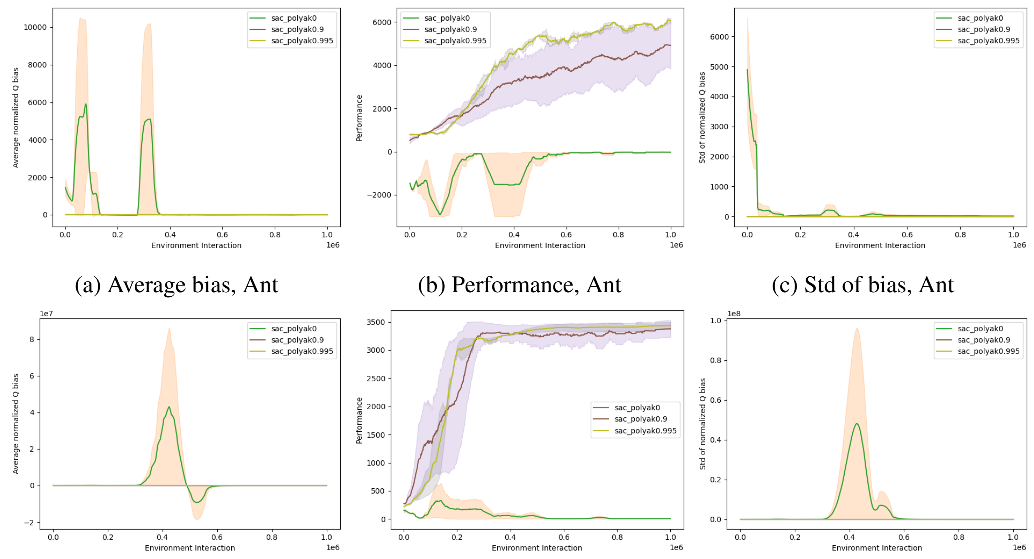
In this section the experimental results for REDQ with SAC setting is provided for two challenging MuJoCo environments, Hopper-v2 and Ant-v2. By applying the default hyperparameters suggested in the SAC paper while setting $utd_{ratio} = 1$, $num_Q = 2$, the REDQ implementation is equivalent to running SAC. In addition, $auto_{alpha}$ is set to False to turn down adaptive SAC. Here we provide ablation study for six hyperparamters, and each experiment is run on both environments with two seeds for initialization. At the beginning of the training 5000 random data points are collected, then 1000 epochs are run with 1000 data points generated for each epoch, so the total data points the agent used is 1005000. We use three criteria to value the agent under the different hyperparameter settings: the average normalized Q value bias, the std of that normalized bias and the performance of the agent. The Q value bias is exactly the way it is defined in the methods section, and both the average bias and its std are obtained during training. After each epoch, a test episode is run with the current policy, and we record the undiscounted sum of all the rewards in that episode as the performance of the agent.

4.1 EXPERIMENT GROUP 1.1

In experiment group 1.1 we first test the influence of polyak and α on training, then we test the difference between deterministic or stochastic action selection during evaluation. Polyak is the update ratio of the target networks that computes Q value estimate and alpha control the extent of entropy exploration during training. The detailed values of these hyperparameters is shown in table1.

Variant name	Variant value
polyak	0
	0.9
	0.995
alpha	0
	0.01
	0.3
	1
action-selection	deterministic
	stochastic

Table 1: Variant values of experiment group 1.2



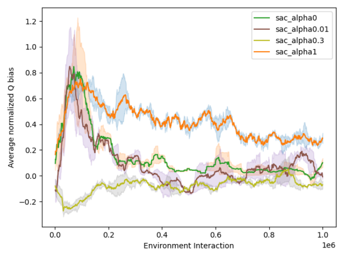
(a) Average bias, Ant (b) Performance, Ant (c) Std of bias, Ant

(d) Average bias, Hopper (e) Performance, Hopper (f) Std of bias, Hopper

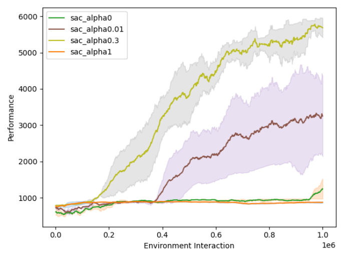
Figure 2: results of different polyak implementations in SAC setting

Figure 2 shows the difference between polyak values of 0, 0.9 and 0.995 during training. The SAC agent achieved similar results in Ant and Hopper when the polyak value is the same. In both environments, polyak value of 0 shows consistently low performance around 0, high average bias and high std of that bias in the form of narrow spikes. The agent shows no sign of learning because the target networks that perform Q value estimation are never updated. When polyak is set to 0.9 or 0.995, the agent behaved similarly with very steady low bias and bias std, and their curves are overlapped in the corresponding two graphs. Polyak value of 0.9 learned relatively faster at the beginning, but the agent with polyak = 0.995 is the one that achieves the highest performance in the long run. The higher the polyak value, the more of the old target networks are retained during the updates, and the high performance may be attributed to the networks' insensitivity to the occurrence of noisy recent data.

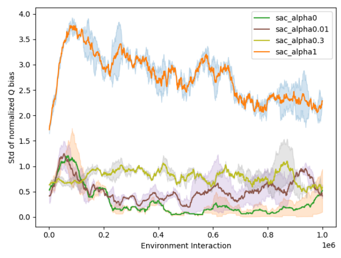
Figure 3 shows the difference between alpha values of 0, 0.01, 0.3 and 1 during training. α is a weight assigned to the entropy and it balances the exploration and exploitation of the agent. The higher α, the more emphasis is placed on exploration. $\alpha = 0$ means no entropy is involved to encourage exploration, and due to that the agent gives consistent low performance, high bias and high bias std across both Ant and Hopper. $\alpha = 0.3$ and $\alpha = 0.01$ achieves the best and the second best performance in both environments, and the difference is considerably larger in Ant than in



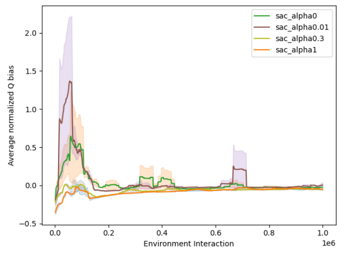
(a) Average bias, Ant



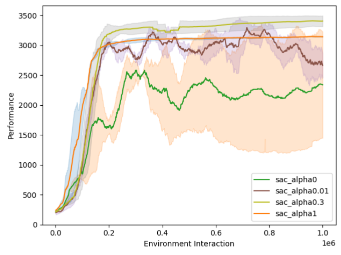
(b) Performance, Ant



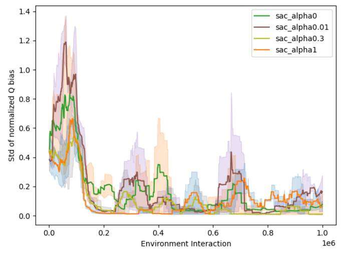
(c) Std of bias, Ant



(d) Average bias, Hopper



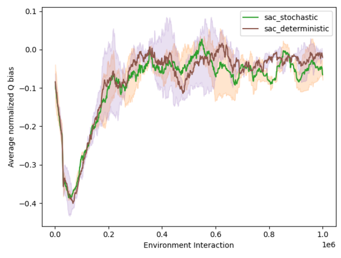
(e) Performance, Hopper



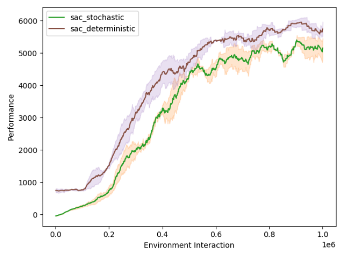
(f) Std of bias, Hopper

Figure 3: results of different alpha implementations in SAC setting

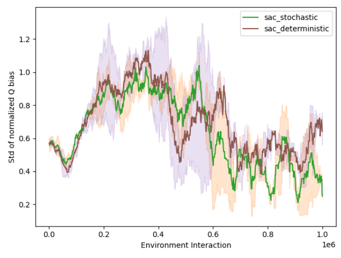
Hopper, because Ant is relatively more challenging. This also leads to the difference of average bias and bias std under $\alpha = 1$ between the two environments, where the agent suffers from the highest bias and bias std in Ant while in Hopper the bias and bias std are the lowest. In correspondence, $\alpha = 1$ achieves good performance in Hopper but poor performance in Ant. The experiment hints that larger α is more suitable for less challenging environments, and lower alpha is more suitable challenging environments, but to verify this more comprehensive analysis is required.



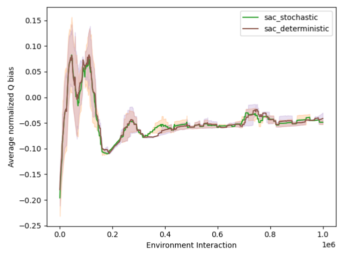
(a) Average bias, Ant



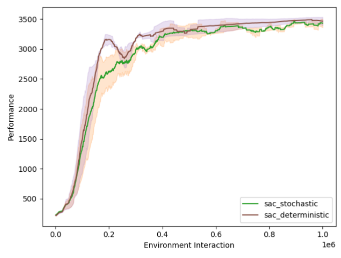
(b) Performance, Ant



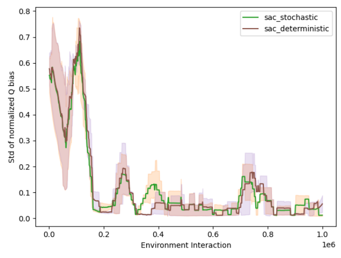
(c) Std of bias, Ant



(d) Average bias, Hopper



(e) Performance, Hopper



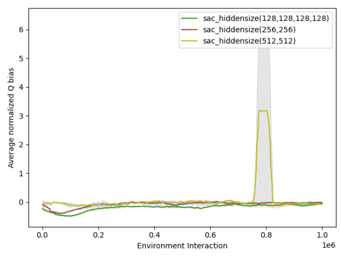
(f) Std of bias, Hopper

Figure 4: results of different alpha implementations in SAC setting

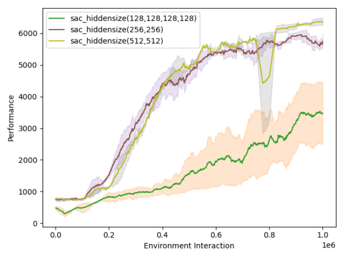
Figure 4 shows the difference between deterministic action selection and stochastic action selection during agent evaluation that occurs once every epoch. In deterministic setting the best action is chosen, and in stochastic setting the action is chosen based on the Gaussian probability distribution over the whole action space. The average bias and bias std is highly similar in both setting across the two environments, because these data is generated during training instead of evaluation, and it is only nature that the curves look identical. For agent performance, deterministic action selection yields better results, and the difference is again larger in Ant than in Hopper because Ant is more challenging. The update of the action space Gaussian distribution during training may have reduced the difference between deterministic and stochastic action selection during evaluation, but this calls for further examination.

4.2 Experiment Group 1.2

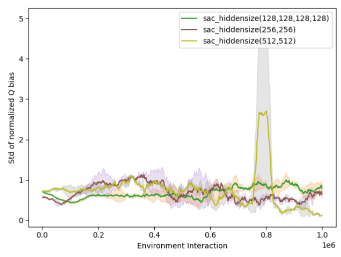
In experiment group 1.2 we test the influence of network structure, replay buffer size and batch size on training. Network structure involves the width and depth of the network hidden size that affects network performance, replay buffer size determines how much old data is stored for updating, and batch size determines how many data points are used per update. The detailed values of these hyperparameters is shown in the runtime table at the end of 4.2.



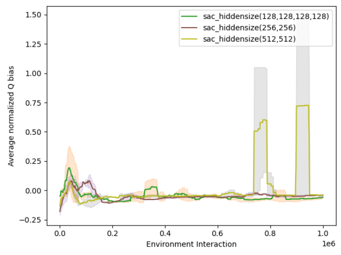
(a) Average bias, Ant



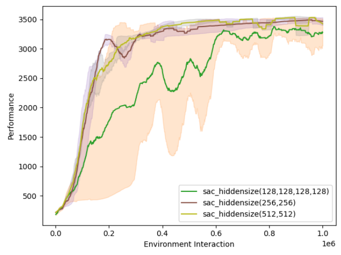
(b) Performance, Ant



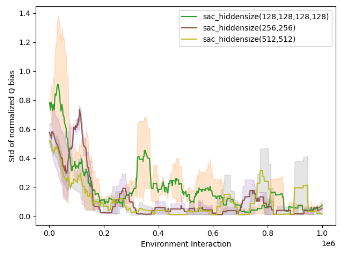
(c) Std of bias, Ant



(d) Average bias, Hopper



(e) Performance, Hopper

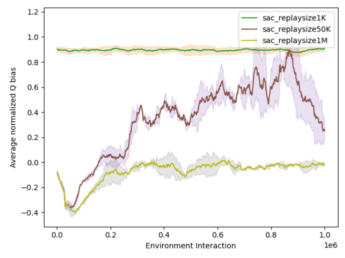


(f) Std of bias, Hopper

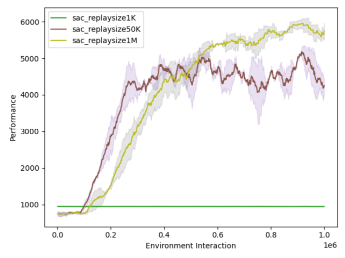
Figure 5: results of different hidden size implementations in SAC setting

Figure 5 shows the difference between different network structures of (128,128,128,128), (256,256) and (512,512) during training. From average bias and bias std perspective, all the network structures managed to keep them low, except that the widest network of (512,512) gives occasional spikes of bias and bias std during training. From performance perspective, networks of (256,256) and (512,512) behaved similarly with the wider network of (512,512) performing slightly better, while the narrower and deeper structure of (128,128,128,128) performed worse. The distinction between Ant and Hopper again caused the performance difference to be larger in Ant than in Hopper. The experiment demonstrates that a network of width 256 is sufficient compared to these two other settings, and we should not sacrifice network width for depth, but this requires further experiments to verify.

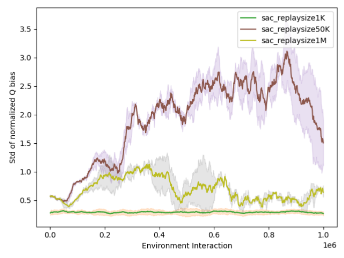
Figure 6 shows the difference between different replay buffer sizes of 1K, 50K and 1M during training. The distinction between buffer sizes of 50K and 1M is consistent in both environments. The agent learns more quickly under 50K buffer size and achieves higher performance in early phase of training, but the highest performance at the end is achieved when buffer size is 1M. Buffer size of 50K also produces higher average bias and bias std. For agent with 1K buffer size, in both cases the performance of the agent is rather poor due to it experiencing no training, because in the code base we set a bar on how much data we must possess to perform training and updating. In addition, the bias std is also similar, but the average bias of the agent is larger in Ant than in Hopper, again due to the difference between the environments. The experiment shows that buffer size of 1M appears most ideal, but this calls for further examination.



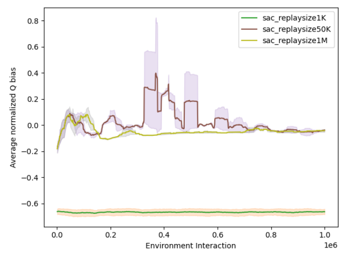
(a) Average bias, Ant



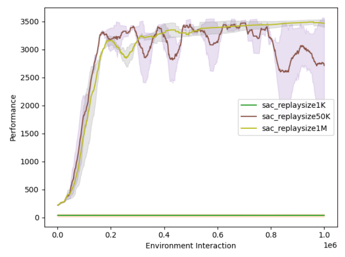
(b) Performance, Ant



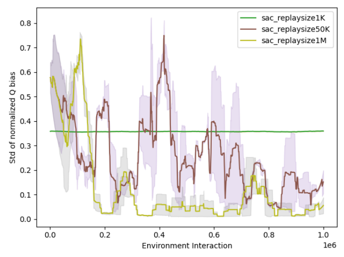
(c) Std of bias, Ant



(d) Average bias, Hopper

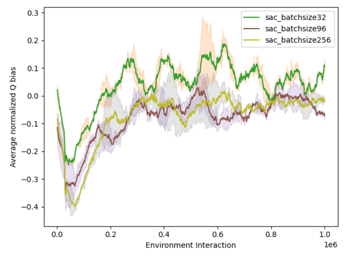


(e) Performance, Hopper

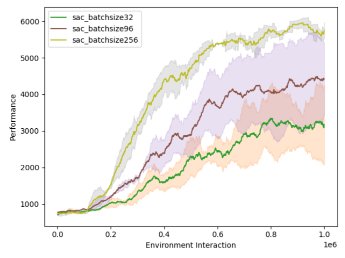


(f) Std of bias, Hopper

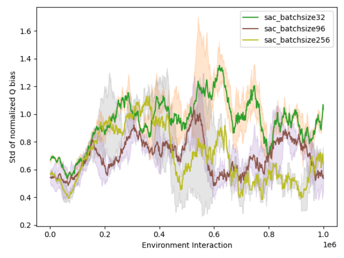
Figure 6: results of different replay size implementations in SAC setting



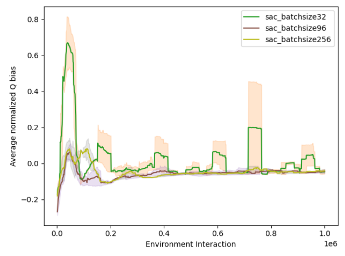
(a) Average bias, Ant



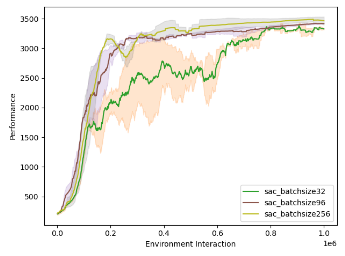
(b) Performance, Ant



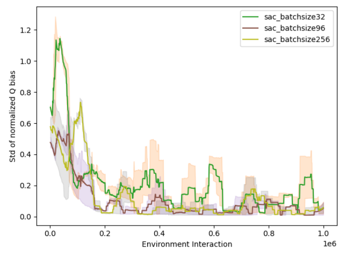
(c) Std of bias, Ant



(d) Average bias, Hopper



(e) Performance, Hopper



(f) Std of bias, Hopper

Figure 7: results of different batch size implementations in SAC setting

Figure 7 shows the difference between different batch size of 32, 96 and 256 during training. The comparison results is similar across all three criteria and both environments. In all the cases, 256 batch size gives the highest performance , lowest average bias and generally lowest bias std. batch size of 96 produces similar average bias and bias std compared to batch size 256, while there is a gap in the performance in Ant but not in Hopper. bath size of 32 produces worst average bias and worst bias std, and its performance is also the lowest. For all three criteria, the difference between different batch sizes is larger in Ant than in Hopper due to Ant being more challenging. The batch size of 256 appears as the best choice, but for environments that are not very demanding a batch size of 96 appears sufficient.

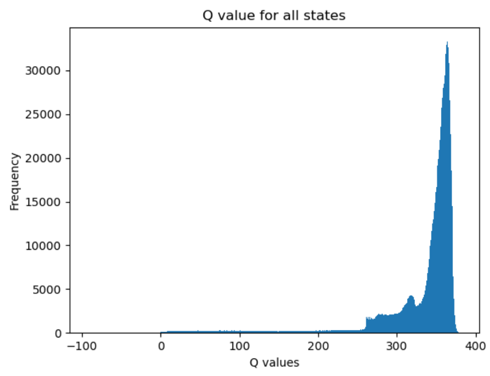
Variant name	Variant value	Runtime in Ant-v2	Runtime in Hopper-v2
network hidden size	(128,128,128,128)	6.59	5.51
	(256,256)	7.84	5.32
	(512,512)	16.23	17.05
replay buffer size	1K	0.96	0.30
	50K	7.62	6.09
	1M	6.42	5.31
batch size	32	4.32	3.64
	96	4.94	4.27
	256	6.68	5.42

Table 2: Computation time table of experiment group 1.2

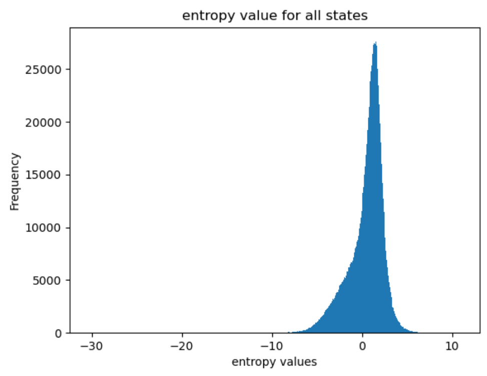
Table 2 shows the total runtime of training of the experiment group 1.2 in hours. The numbers are the average result of the two seed runs. In general, Ant as a more challenging environments takes more time to run than Hopper under the same parameter setting. For the structure of the neural network, the width of the network greatly influences the runtime as is shown between (256,256) and (512,512) network structure, whereas a deeper but narrower structure of (128,128,128,128) uses similar runtime compared with default setting (256,256). For replay buffer size, 1K only requires small amount of runtime compared with other settings in the experiment, and a counter intuitive result appears in the comparison between 50K and 1M replay buffer size. 50K buffer size actually is more runtime-demanding, and this result may come from the fact that we only use two seeds. It could also be that the update of replay buffer contributes to the difference. Since we obtain 1000 data points per epoch, 1M buffer size means we almost never discard old data from the buffer, but that would often happen with a 50K-sized buffer. For batch size, a steady increase can be observed with respect to the increase in the batch size, since larger batch size means more sampling time is taken during training. To conclude, wider network structure, intermediate replay buffer size and larger batch size contributes to the increase in runtime.

4.3 Experiment Group 2.1

In experiment group 2.1 we visualize the estimated Q values and the entropy values given by a trained agent of all the states by plotting two histograms. The x-axis denotes the corresponding Q or entropy values and the y-axis denote the frequency of the values.



(a) Average bias, Ant



(b) Performance, Ant

Figure 8: results of different alpha implementations in SAC setting

The left graph of figure 8 shows the estimated Q value of all the states. We can observe that the highest frequency of Q estimates occurs between 300 and 400. This number seems very different from the performance of the agent under all different hyperparameter settings in the previous figures. Note that the performance is recorded as the undiscounted sum of all the rewards, while an Q estimate is a discounted sum of all current and future rewards. Since we set total epoch to be 1000 and we apply discount factor of 0.99, once we discount the rewards in the performance, we would obtain a number that agrees with the scale of our Q estimate, thus the Q estimate here and the agent performance is consistent. The reason why there should be very few states of low Q estimate is because these are the states that are very close to termination and very little future reward is involved, and the Q value distribution shown in the graph reflects this.

The right graph of figure 8 shows the entropy value of all the states. Since we view that every state has its own Gaussian distribution of actions, there should be variance among the entropy of all the states, and the graph reflects this. If an universal std instead of entropy is used to encourage exploration, then the histogram should only involve a straight line. The larger the standard deviation of Q bias of the states, the larger the entropy would be, which means the more need for exploration in that state.

4.4 Result Summary

The analysis we perform in experiment group 1.1 is similar to the one done in Henderson et al. (2019) as the same three hyperparameters are tested, except that we use the SAC setting of REDQ, and we run additional experiment group 1.2 and we perform additional analysis on Q bias and bias std. However, the experiments conducted in Henderson et al. (2019) is relatively more robust, and this project suffers from the problems with too few random seeds and hyperparameter value comparison. The more influential hyperparameters of the six examined are α, polyak and batch size, but in actual implementation the environment would greatly impact the choice of hyperparameters. There is no direct correlation between performance and average Q bias, as high performance can occur in both high average bias and low average bias case. The reason behind this is that when the bias is similar and of the same direction, it would not affect the relative choice of action very much, and that could still lead to high performance. It is the combination of average bias and bias std that influences the action selection during training.

5 Conclusion

Using carefully designed ablation study experiments, we implement REDQ with SAC setting and test the influence of six hyperparameters on the training of the agent for two MuJoCo environments of Ant and Hopper. The result indicates that the two MuJoCo environments reacts differently to some of the hyperparameters tested, and hyperparametering tuning should be specified with respect to the environment to achieve best agent performance. More challenging environments like Ant demands more complex algorithm setting for better optimization. α, polyak and batch size are relatively more influential in both environments. We verify that the agent performance agrees with the Q estimate, and the distribution of Q estimate and entropy value is reasonable across all the states. A possible future work direction is to run experiments like hyperparameter grids that are more comprehensive than current ablation study to determine a set of ideal hyperparameters, so that we take into account the interaction between different hyperparameters. Moreover, more seeds could be applied to better reflect the agent's average performance, larger set of values could be used to further examine the pattern between hyperparameter scales and agent performance, and more setting of the REDQ other than SAC version could be applied.

References

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

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods, 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.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters, 2019.

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, 2019.