DEEP REINFORCEMENT LEARNING COURSE PROJECT

Alison Yao (yy2564)
Data Science
NYU Shanghai

ABSTRACT

The overestimation bias is a major problem in many DRL algorithms. The clipped double Q method has proven to be effective in bias reduction for SAC, but in this report, an alternative called the multi-step method is tested to see if it can be equally effective. The report goes on to explore if the collective effect of clipped double Q and multi-step method on SAC can further reduce the bias. Imperially, we find that the default SAC algorithm, which uses the clipped double Q method, remains the best.

1 INTRODUCTION

Overestimation biases arise when Q values are overestimated due to the maximization of a noisy value estimate. Consequences of overestimation biases include overestimation of bad states, suboptimal policy updates and failure of convergence. Fujimoto et al. (2018) show that such overestimation problem extends from discrete action space settings to continuous action space settings.

This paper focuses on two proposed solutions, namely the clipped double Q method and the multi-step method. The former is a major advance from DDPG to TD3 in overestimation reduction (Fujimoto et al., 2018); the latter also mitigates the overestimation problem for DDPG by exploiting bootstrapping and can achieve comparable performance to TD3 (Meng et al., 2020). However, the comparative and collective effects of these two methods on SAC have not been studied.

In this paper, we will test the overestimation reduction performance of clipped double Q method and multi-step method on the base algorithm of SAC. More specifically, we will run experiments on OpenAI Gym to investigate three questions empirically:

- the positive effect of clipped double Q in SAC compared to SAC without clipped double Q
- whether multi-step method can achieve the same positive effect of clipped double Q
- whether we can use double Q and multi-step methods together to further boost performance and lower training time

2 RELATED WORK

The base algorithm of this study is Soft Actor-Critic (Haarnoja et al., 2018), an off-policy actor-critic algorithm that combines maximum entropy reinforcement learning framework with TD3. As mentioned in the introduction, the clipped double Q method is integrated into SAC to reduce overestimation.

The clipped double Q method proposed by Fujimoto et al. (2018) is an effective universal overestimation reduction method theoretically and experimentally for both deep Q-learning with discrete actions and actor-critic settings. Clipped double Q method can reduce overestimation bias because it chooses the smaller Q value from the two Q functions to calculate the target so that the value target is less than or at least equal to using the standard Q-learning target (Fujimoto et al., 2018).

The multi-step method integrates multi-step bootstrapping and enables fast propagation of knowledge about the outcomes of selected actions by taking into account the rewards n steps forward. Meng et al. (2020) proposed two multi-step methods on DDPG, which are Multi-step DDPG (MDDPG) and Mixed Multi-step DDPG (MMDDPG). The step sizes in MDDPG are manually set while

MMDDPG takes the average of different step sizes to update the target of Q function and is therefore more stable than MDDPG. MDDPG and MMDDPG show comparable final performance and learning speed as TD3 on most tasks and MMDDPG can even outperform TD3 occasionally (Meng et al., 2020). In this paper, we focus on the multi-step method in MDDPG.

Our method of study is similar to the experiments in Fujimoto et al. (2018) and Meng et al. (2020). However, we use SAC Haarnoja et al. (2018) as the base algorithm instead of DDPG. We will go on to investigate the respective effects and the collective effect of clipped double Q and multi-step methods on SAC.

3 METHODS

The overestimation bias is estimated by the Q estimate minus the Monte Carlo return for each state-action pair. Both the Q estimate from the value network(s) and the Monte Carlo return (MC return) are approximations of the true q value. The Q estimate is biased but the MC return is unbiased. Therefore, the difference between them can estimate the overestimation bias.

We conduct experiments on 4 SAC variants. Besides default SAC, the SAC without clipped double Q method is denoted as SAC-Single. Adding the multi-step method to SAC and SAC-Single, we have MSAC(k) and MSAC-Single(k) where k is the number of steps look-ahead in target calculation. When k = 1, SAC is the same as MSAC(1) and SAC-Single is the same as MSAC-Single(1). Table 1 illustrates the similarities and differences of the four SAC variants. When use_single_variant is False, the clipped double Q method is used. For the choice of k, we run k = 5 for MSAC and k = 2,5,8 for MSAC-Single. Therefore, this paper has SAC, SAC-Single, MSAC(5), MSAC-Single(2), MSAC-Single(5) and MSAC-Single(8) 6 different sets of tests in total.

Table 1: SAC Variant Comparison

Algorithm Name	use_single_variant	Multi-step
SAC	False	1 step
SAC-Single	True	1 step
MSAC(k)	False	k steps
MSAC-Single(k)	True	k steps

The Q targets of different SAC variants are computed differently. In SAC, the clipped double Q method requires two value networks and takes the minimum of the two Q estimates to calculate the target. The formula is given below:

$$y = r + \gamma (1 - d) \left(\min_{i=1,2} Q_{\phi_{\text{targ},i}} (s', \tilde{a}') - \alpha \log \pi_\theta (\tilde{a}' \mid s') \right), \quad \tilde{a}' \sim \pi_\theta (\cdot \mid s') \tag{1}$$

where s' denotes the next state and d denotes the done signal.

SAC-Single, on the other hand, only has one value network. The target formula is given below:

$$y = r + \gamma (1 - d) \left(Q_{\phi_{\text{targ},1}} (s', \tilde{a}') - \alpha \log \pi_\theta (\tilde{a}' \mid s') \right), \quad \tilde{a}' \sim \pi_\theta (\cdot \mid s') \tag{2}$$

where s' denotes the next state and d denotes the done signal.

MSAC(k) and MSAC-Single(k) add multi-step bootstrapping to the target so the reward r becomes the sum of k-step discounted reward and weight of Q or Q_{min} and entropy terms becomes γ^k. Therefore, the target formula for MSAC(k) and MSAC-Single(k) are:

$$y = \sum_{j=0}^{k-1} \gamma^j r_{t+j} + \gamma^k (1 - d) \left(\min_{i=1,2} Q_{\phi_{\text{targ},i}} (s', \tilde{a}') - \alpha \log \pi_\theta (\tilde{a}' \mid s') \right), \quad \tilde{a}' \sim \pi_\theta (\cdot \mid s') \tag{3}$$

$$y = \sum_{j=0}^{k-1} \gamma^j r_{t+j} + \gamma^k (1 - d) \left(Q_{\phi_{\text{targ},1}} (s', \tilde{a}') - \alpha \log \pi_\theta (\tilde{a}' \mid s') \right), \quad \tilde{a}' \sim \pi_\theta (\cdot \mid s') \tag{4}$$

where s' denotes the state $t + k$ and d denotes the done signal for both (3) and (4).

The pseudocode for SAC, MSAC and MSAC-Single are as follows. Please note that the policy update part for single network and double networks are slightly different.

Algorithm 1 Soft Actor-Critic (SAC)

Initialize policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$. Set target parameters $\phi_{\text{targ},i} \leftarrow \phi_i$, for $i = 1, 2$
repeat
  Take one action $a_t \sim \pi_\theta(\cdot|s_t)$. Observe reward r_t, new state s_{t+1}.
  Add data to buffer: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r_t, s_{t+1}, d)\}$
  Sample a mini-batch $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
  Compute the Q target y (same for both Q-functions):

$$y = r + \gamma\,(1-d)\left(\min_{i=1,2} Q_{\phi_{\text{targ},i}}\left(s', \tilde{a}'\right) - \alpha \log \pi_\theta\left(\tilde{a}' \mid s'\right)\right), \quad \tilde{a}' \sim \pi_\theta\left(\cdot \mid s'\right)$$

  where s' denotes the next state and d denotes the done signal
  for $i = 1, 2$ **do**
    Update ϕ_i with gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s,a) - y\right)^2$$

  Update target networks with $\phi_{\text{targ},i} \leftarrow \rho \phi_{\text{targ},i} + (1-\rho)\phi_i$
  Update policy parameters θ with gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \left(\min_{i=1,2} Q_{\phi_i}\left(s, \tilde{a}_\theta(s)\right) - \alpha \log \pi_\theta\left(\tilde{a}_\theta(s)|s\right)\right), \quad \tilde{a}_\theta(s) \sim \pi_\theta(\cdot \mid s)$$

Algorithm 2 Multi-Step Soft Actor-Critic (MSAC)

Initialize policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$. Set target parameters $\phi_{\text{targ},i} \leftarrow \phi_i$, for $i = 1, 2$. Set k manually (eg: k=2,5,8)
repeat
  Take one action $a_t \sim \pi_\theta(\cdot|s_t)$. Observe reward r_t, new state s_{t+1}.
  Add data to buffer: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r_t, s_{t+1}, d)\}$
  Sample a mini-batch $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
  Compute the Q target y (same for both Q-functions):

$$y = \sum_{j=0}^{k-1} \gamma^j r_{t+j} + \gamma^k\,(1-d)\left(\min_{i=1,2} Q_{\phi_{\text{targ},i}}\left(s', \tilde{a}'\right) - \alpha \log \pi_\theta\left(\tilde{a}' \mid s'\right)\right), \quad \tilde{a}' \sim \pi_\theta\left(\cdot \mid s'\right)$$

  where s' denotes the state $t + k$ and d denotes the done signal
  for $i = 1, 2$ **do**
    Update ϕ_i with gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s,a) - y\right)^2$$

  Update target networks with $\phi_{\text{targ},i} \leftarrow \rho \phi_{\text{targ},i} + (1-\rho)\phi_i$
  Update policy parameters θ with gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \left(\min_{i=1,2} Q_{\phi_i}\left(s, \tilde{a}_\theta(s)\right) - \alpha \log \pi_\theta\left(\tilde{a}_\theta(s)|s\right)\right), \quad \tilde{a}_\theta(s) \sim \pi_\theta(\cdot \mid s)$$

Algorithm 3 Multi-step Soft Actor-Critic Single (MSAC-Single)

1: Initialize policy parameters θ, Q-function parameters ϕ_1, empty replay buffer $\mathcal{D}$. Set target parameters $\phi_{\text{targ},1} \leftarrow \phi_1$. Set k manually (eg: k=2,5,8)
2: **repeat**
3: Take one action $a_t \sim \pi_\theta(\cdot|s_t)$. Observe reward r_t, new state s_{t+1}.
4: Add data to buffer: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r_t, s_{t+1}, d)\}$
5: Sample a mini-batch $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
6: Compute the Q target y (only one Q-function):

$$y = \sum_{j=0}^{k-1} \gamma^j r_{t+j} + \gamma^k \left(Q_{\phi_{\text{targ},1}} \left(s', \tilde{a}' \right) - \alpha \log \pi_\theta \left(\tilde{a}' \mid s' \right) \right), \quad \tilde{a}' \sim \pi_\theta \left(\cdot \mid s' \right)$$

where s' denotes the state $t + k$ and d denotes the done signal
7: Update ϕ_1 with gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_1}(s, a) - y \right)^2$$

8: Update target networks with $\phi_{\text{targ},1} \leftarrow \rho \phi_{\text{targ},1} + (1 - \rho)\phi_1$
9: Update policy parameters θ with gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \left(Q_{\phi_1} \left(s, \tilde{a}_\theta(s) \right) - \alpha \log \pi_\theta \left(\tilde{a}_\theta(s) | s \right) \right), \quad \tilde{a}_\theta(s) \sim \pi_\theta(\cdot \mid s)$$

4 RESULTS

The 6 sets of experiments, SAC, SAC-Single, MSAC(5), MSAC-Single(2), MSAC-Single(5) and MSAC-Single(8), are conducted on two environments in OpenAI Gym, Ant-v2 and Hopper-v2, with random seeds of 0, 1, 2, 3, making up 6 * 2 * 4 = 48 experiments in total. For each experiment, the number of data-points is 1,000,000. Henderson et al. (2018) have pointed out valid concerns in reproducibility in DRL experiments. Random seeds, for one, introduce non-determinism and can potentially result in misleading outcomes. Therefore, in this paper, multiple runs on four seeds 0,1,2,3 are averaged to alleviate the high variance problem across trails and random seeds. Experiment results are presented in Fig 1 and Fig 2 to answer the three questions proposed in the Introduction section.

When clipped double Q is removed from SAC, SAC becomes SAC-Single. We can see from Fig 1 that the average test return, average Q value rises and reaches a higher peak at the beginning and the average Q bias increases, so SAC-Single has an overall worse performance than SAC. Therefore, clipped double Q is an effective component in SAC that reduces overestimation bias and divergence.

To compare the performance of the clipped double Q method and the multi-step method, we can see SAC and MSAC-Single(5) in Fig 1. MSAC-Single(5) has a similar average Q bias as SAC and has a slightly lower divergence risk than SAC, but it cannot compete with SAC in average test return, or performance. Therefore, we cannot necessarily replace the clipped double Q method with the multi-step method.

To see the collective effect of the clipped double Q method and the multi-step method, we can compare SAC with MSAC(5) in 1. Again, MSAC(5) has a similar average Q bias as SAC and has a slightly lower divergence risk than SAC, but it has a much lower average test return, or performance, than SAC. Therefore, we believe we cannot use the clipped double Q method and the multi-step method at the same time.

However, it does not mean the multi-step method is completely gratuitous. Judging from Fig 2, adding multi-step to SAC-Single has a significant improvement in average test return/performance, has a lower divergence risk and has a lower Q bias. Interestingly, regarding the choice of multi-step k, k = 2 seems to be the best. k = 2 has the highest performance in Ant-v2 and the second-highest performance in Hopper-v2 in the end and its Q bias is the closest to 0. As k increases, the more

underestimated Q estimate is, but it is difficult to conclude how the performance changes because Hopper-v2 is more unstable than Ant-v2 and they have very different graphs.

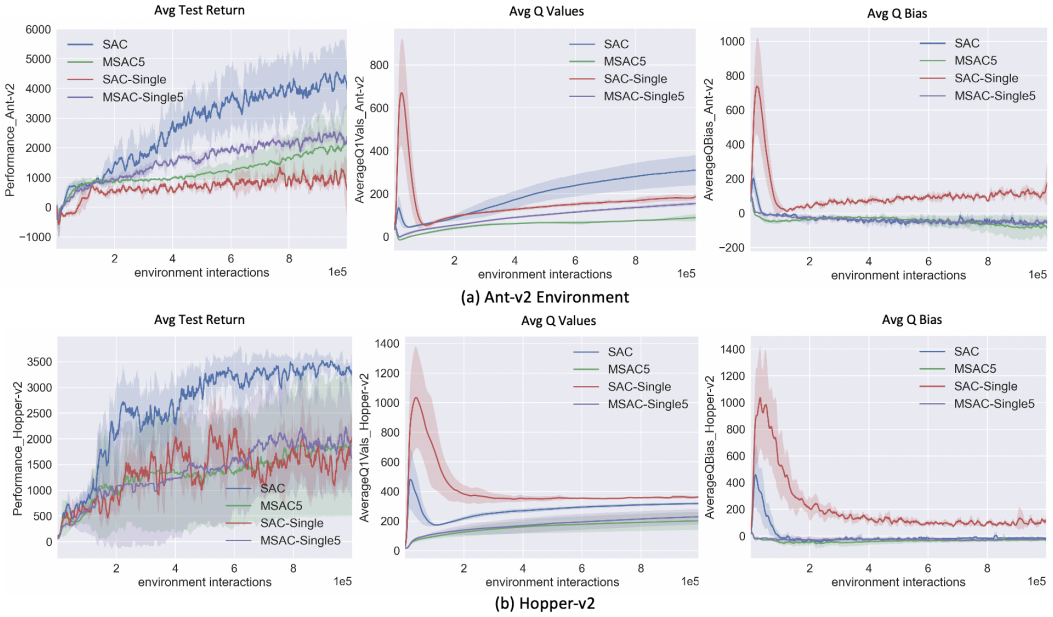

Figure 1: Comparison of performance, average Q value and average Q bias of SAC, MSAC(5), SAC-Single and MASC-Single(5) in Ant-v2 and Hopper-v2.

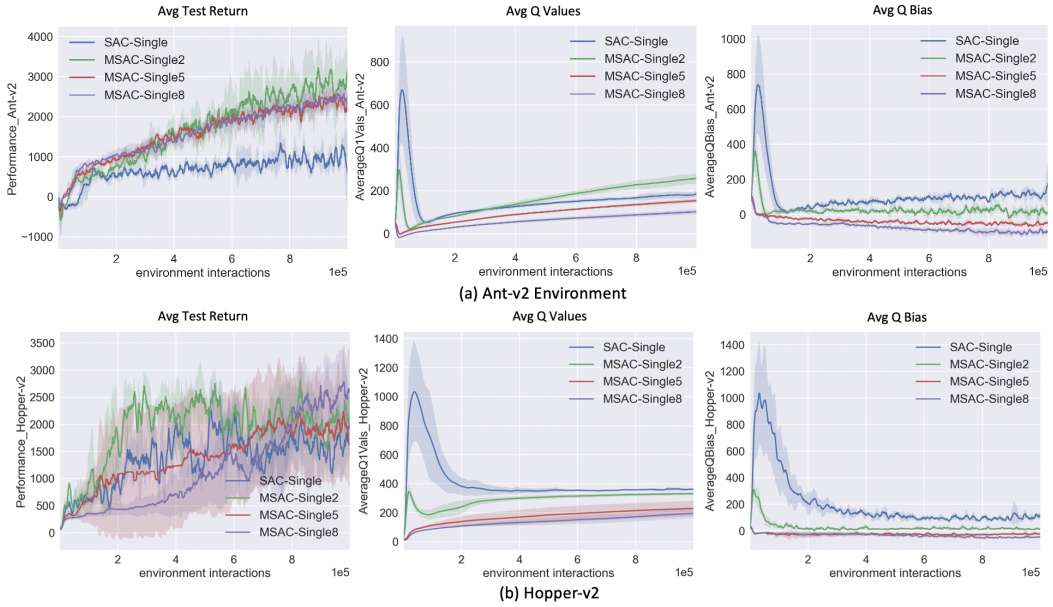

Figure 2: Comparison of performance, average Q value and average Q bias of SAC-Single, MASC-Single(2), MASC-Single(5) and MASC-Single(8) in Ant-v2 and Hopper-v2.

In terms of computation time, when we apply multi-step to SAC, it does not slow down the computation a lot, only by less than 10%.

5 CONCLUSIONS

In this paper, we study empirically the comparative and collective effects of the clipped double Q method and the multi-step method. We find that SAC with clipped double Q still outperforms SAC-Single, MSAC(5) and MSAC-Single(5). Replacing or combining the clipped double Q method with multi-step method does not yield a better average test return, or performance, despite a minor edge of divergence reduction. But the multi-step method does improve SAC-Single in general, with the multi-step k performing the best.

It is reassuring to know that TD3 is indeed a state-of-the-art algorithm with very strong performance. It is also surprising to find out that sometimes the positive effects of two methods counteract. Therefore, in the future, we think it would be helpful to investigate why the clipped double Q method and the multi-step method counteract so that we might be able to combine the two in a more elegant way to yield better results. In addition, the effect of the multi-step k is still unclear. We may experiment with more k's and much larger k's to see if the performance will improve.

6 ADDITIONAL EXPERIMENTS AND DISCUSSIONS

6.1 RENDERING THE SAC AGENT

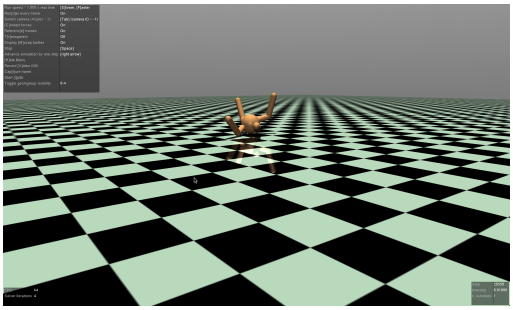

Figure 3: Ant-v2 Untrained Agent

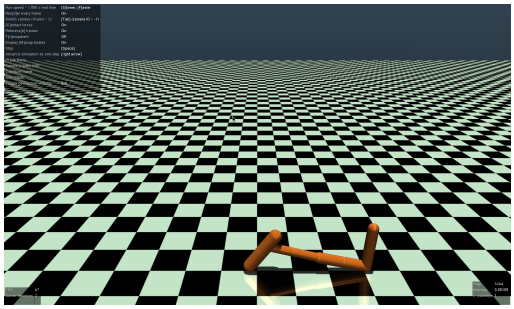

Figure 4: Hopper-v2 Untrained Agent

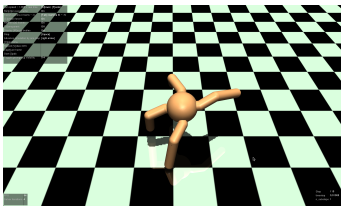

Figure 5: Ant-v2 Trained Agent

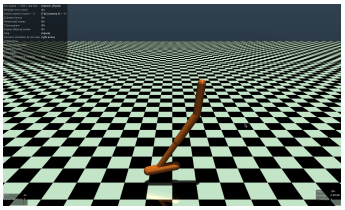

Figure 6: Hopper-v2 Trained Agent Hopping

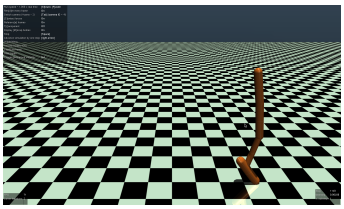

Figure 7: Hopper-v2 Trained Agent Landing

The untrained agents take actions randomly, so neither ant nor hopper seems to have a goal in mind when moving around and this is completely expected. The trained agents perform much better than the untrained ones. The ant is much less likely to flip over after training and it sticks to moving toward one direction instead of crawling all over the place. If it does eventually flip over, it takes a much longer time to do that. The trained hopper is showing great performance improvement in hopping and not falling. It can actually hop and land as shown in Fig 6 and Fig 7. The untrained hopper always immediately falls down once an episode starts.

REFERENCES

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.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Lingheng Meng, Rob Gorbet, and Dana Kulić. The effect of multi-step methods on overestimation in deep reinforcement learning. *arXiv preprint arXiv:2006.12692*, 2020.

A HYPER-PARAMETERS AND IMPLEMENTATION DETAILS

Table 2: List of training hyper-parameters

Hyper-parameter	**Value(s)**
number of bias calculation episodes	5
hidden sizes	[256, 256]
seeds	0, 1, 2, 3
steps_per_epoch	1000
number of training epoch	1000
max length of replay buffer	1,000,000
discount factor	0.99
polyak	0.995
learning rate	0.0003 (auto tune)
entropy regularization coefficient	0.2
batch size	256
steps for uniform-random action selection	10,000
max episode length	1000
save frequency	1

To compute the bias, we run N=10 episodes on the current policy. For each data-point in an episode, we store the state, the action and the reward in separate lists. To compute the MC return, we loop the reward list backward to calculate the returns. Then we reverse the return list in the correct order so all state-action pairs and returns are matched. After the end of an episode, we have four lists, the states, the actions, the rewards and the returns. we select the valid data-points and store the states, the actions and the returns in bigger lists which act as buffers. After all 10 episodes are done, we convert the bigger state list and the bigger action list into tensors and put them through the Q network(s) to compute the Q estimate. The MC returns are already stored in the bigger MC return list, so all that is left is converting it into a tensor and calculate the difference between the Q and MC to get the bias.

We deal with episode termination by setting a max buffer size, even though we are not using a buffer directly. If an episode terminates before 1000 steps, all the data-points in this episode are valid. If it does not terminate after 1000 steps, we truncate the episode and only select the first 800 data-points as the valid ones, because the later 200 data-points do not have correct MC return values. The reason is that when an episode is truncated before actual termination, rewards after 1000 are not added, making the MC return lower than the actual value. The first few hundred steps are less affected than the last few hundred steps because of the discount factor. For example, when the 1001 step is taken, G_t (t between 0 and 1000) grows by $0.99^{1000-t}r_{1001}$. The earlier the time step is, which means the smaller t is, the less affected its MC return is. And 0.99^{200} (t = 800) is around 0.134, so the effect of $0.134r_{1001}$ is already fairly small, thus we choose the first 800 points as valid points.

The policy update part for SAC and SAC-Single is different because there is only one Q network for SAC-Single. We choose q1 as the single network, so in this case, we do not need to take the minimum of Q1 and Q2 anymore.