Reinforcement Learning
Assignment 3: Advanced Actor Critic - SAC

Daniël Zee (s2063131) [1] **Simon Klaver (s1140760)** [1]

Abstract

In this assignment, we studied the Soft Actor Critic (SAC) deep reinforcement learning algorithm and tested its performance on the CartPole-v1 environment. For the implementation we utilize twin critic networks with a target network for each, entropy regularization, and off-policy learning through the use of a replay buffer. In order to make SAC applicable to discrete action spaces, we devised both expected and sampled update strategies for both the actor and critic networks. Experiments show the temperature (α) significantly impacts performance, and clipped double Q-learning vastly outperforms the single Q-learning approach, particularly when combined with expected updates rather than sampled ones.

1. Introduction

Previously, we have studied the deep reinforcement learning (RL) algorithms DQN (Mnih et al., 2015), REINFORCE (Williams, 1992) and basic Actor-Critic methods (Konda & Tsitsiklis, 1999). While doing so, we have encountered several important design choices that differentiate deep RL algorithms, influencing their performance and use cases. Fist of all, DQN and REINFORCE only learn the value function and policy function respectively, while Actor-Critic methods combines the two components: an actor, which decides what actions to take, and a critic, which evaluates the quality of those actions. Secondly, there is the classification between on-policy and off-policy algorithms. DQN is off-policy, as it uses experiences obtained from older versions of the policy, stored in a replay buffer, and derives the temporal difference (TD) target using the greedy action over the next state, learning a deterministic policy. REINFORCE and the basic Actor-Critic methods are on-policy, only using experiences and target values derived from the current policy, resulting in higher learning stability but lower sample efficiency. These algorithms learn a stochastic policy.

For this assignment, we have studied an advanced Actor-Critic algorithm that aims to form a bridge between deterministic and stochastic policy optimization: Soft Actor Critic (SAC) (Haarnoja et al., 2018a)(Haarnoja et al., 2018b). SAC learns a stochastic policy in an off-policy way, incorporating several engineering tricks to stabilize the learning process. We have implemented a SAC agent and tested its performance on the CartPole-v1 environment by Gymnasium (Towers et al., 2024). An ablation study was performed on the hyperparameters specific to this algorithm and performance was compared to results from previous assignments.

2. Theory

A central part of any RL algorithm is the balance between exploration and exploitation. While DQN had a hyperparameter responsible for this in the ϵ-greedy policy, REINFORCE and the basic Actor-Critic methods had no such parameter, purely relying on the stochasticity of the policy network. SAC does this using entropy regularization, which trains the policy to maximize a trade-off between expected return and entropy. The entropy over a probability distribution, in our case the action probabilities over a state (1), is a measure of randomness. The higher the entropy, the more uniform the distribution, resulting in more exploration. The entropy term is balanced with the expected return using the trade-off coefficient α (2), also known as the temperature. As both the state values and the entropy term share the expectation over all actions, we can combine them under a single expectation to derive the new value function definition (3).

$$H\left(\pi(s)\right) = \mathop{\mathbb{E}}_{a\sim\pi(\cdot|s)}\left[-\log\pi(a|s)\right] \tag{1}$$

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{a\sim\pi(\cdot|s)}\left[Q^{\pi}(s,a)\right] + \alpha H(\pi(s)) \tag{2}$$

$$= \mathop{\mathbb{E}}_{a\sim\pi(\cdot|s)}\left[Q^{\pi}(s,a) - \alpha\log(\pi(a|s))\right] \tag{3}$$

[1]Leiden Institute of Advanced Computer Science (LIACS), Leiden University, Leiden, The Netherlands. Correspondence to: Daniël Zee <d.f.zee@umail.leidenuniv.nl>, Simon Klaver <s1140760@vuw.leidenuniv.nl>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

For training the critic, SAC uses a variant on Double Q-learning (Hasselt, 2010), called Clipped Double Q-learning (Jiang et al., 2022). In Double Q-learning, two Q networks are trained instead of one, with the goal of counteracting the Q-value overestimation by the network. Clipped Double Q-learning places an upper bound on the Q-value estimates by taking the minimum of the two estimates during both actor and critic updates.

The loss function for the Q-networks (critic) is the mean squared Bellman error (MSBE), minimizing the difference between the TD targets and the current Q estimates. Just like in DQN, the TD targets are computed using target Q-networks. But unlike DQN, where the target Q-network was periodically hard-updated by copying the Q-network parameters, the target Q-networks here are soft-updated by polyak averaging the Q-network parameters over the course of training (4).

$$\phi_{\text{targ},i} \leftarrow (1-\rho)\phi_{\text{targ},i} + \rho\phi_i \tag{4}$$

Another difference compared to DQN is that we calculate our TD targets in an on-policy way, following our current policy network. To do so, we can use the Expected SARSA update rule to calculate the targets (5).

$$y^{\text{expected}}(r,s',d) = r + \gamma(1-d)\cdot \left(\mathop{\mathbb{E}}_{a'\sim\pi_\theta(\cdot|s')}\left[\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s',a') - \alpha\log\pi_\theta(a'|s')\right]\right) \tag{5}$$

This is however requires the full probability distribution over all actions. As this is impossible to obtain for continuous action spaces, we can instead use the SARSA update rule, where we sample a fresh next action $\hat{a}'$ from the current policy (6).

$$y^{\text{sampled}}(r,s',d) = r + \gamma(1-d)\cdot \left(\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s',\tilde{a}') - \alpha\log\pi_\theta(\tilde{a}'|s')\right), \quad \tilde{a}'\sim\pi_\theta(\cdot|s') \tag{6}$$

The off-policy nature of SAC comes from the fact that, just like in DQN, we sample our transitions from a replay buffer $\mathcal{D}$, containing experiences from both the current and older versions of the policy. Putting this together, the MSBE loss function for the Q-networks looks like (7). In practice, we approximate the loss function gradients by sampling a minibatch from the replay buffer, calculating the average (8).

$$L(\phi_i,\mathcal{D}) = \mathop{\mathbb{E}}_{(s,a,r,s',d)\sim\mathcal{D}}\left[\left(Q_{\phi_i}(s,a) - y(r,s',d)\right)^2\right] \tag{7}$$

$$\nabla_{\phi_i}\hat{g}(\phi_i,B) = \frac{1}{|B|}\sum_{(s,a,r,s',d))\in B} \nabla_{\phi_i}\left(Q_{\phi_i}(s,a) - y(r,s',d)\right)^2 \tag{8}$$

The training of the policy network (actor) has the goal of maximizing the objective function, which is now defined as the expected entropy regularized return over all states in the replay buffer (9).

$$J(\theta,\mathcal{D}) = \mathop{\mathbb{E}}_{s\sim\mathcal{D}}\left[\mathop{\mathbb{E}}_{a\sim\pi_\theta(\cdot|s)}\left[\min_{i=1,2} Q_{\phi_i}(s,a) - \alpha\log\pi_\theta(a|s)\right]\right] \tag{9}$$

When calculating the gradient of the objective function with respect to θ (10), we see that the gradients flow through two terms: the weighted sum over the action probabilities and the entropy term. These can both be calculated in the case of a discrete action space. But for continuous action spaces, we can only sample actions, resulting in the gradients not flowing through the first term. The original SAC paper solves this using the reparameterization trick (Haarnoja et al., 2018a), where the policy network instead outputs the mean and standard deviation of a gaussian distribution, used to sample an action. As this approach is not applicable to discrete action spaces (like CartPole-v1), we can not use it here. What we can however do is use the log derivative trick, just like we did for policy gradient methods (Sutton et al., 1999), which allows us to sample actions while still having the gradients flow through both terms of the objective function (11).

Using both versions of the objective function gradients, we can again define two different estimators: one that uses the full expectation over the action probabilities (12), and one that samples a fresh action from the policy, and uses the policy gradient style update (13).

$$\nabla_\theta J(\theta,\mathcal{D}) = \mathop{\mathbb{E}}_{s\sim\mathcal{D}}\left[\sum_a \nabla_\theta\pi_\theta(a|s)\left(\min_{i=1,2} Q_{\phi_i}(s,a) - \alpha\log\pi_\theta(a|s)\right)\right] \tag{10}$$

$$= \mathop{\mathbb{E}}_{s\sim\mathcal{D}}\left[\mathop{\mathbb{E}}_{a\sim\pi_\theta(\cdot|s)}\left[\nabla_\theta\log\pi_\theta(a|s)\left(\min_{i=1,2} Q_{\phi_i}(s,a) - \alpha\log\pi_\theta(a|s)\right)\right]\right] \tag{11}$$

$$\nabla_\theta \hat{g}^{\text{expected}}(\theta, B) = \frac{1}{|B|} \sum_{s \in B} \sum_a \nabla_\theta \pi_\theta(a|s) \left(\min_{i=1,2} Q_{\phi_i}(s,a) - \alpha \log \pi_\theta(a|s) \right) \tag{12}$$

$$\nabla_\theta \hat{g}^{\text{sampled}}(\theta, B) = \frac{1}{|B|} \sum_{s \in B} \nabla_\theta \log \pi_\theta(\tilde{a}|s) \left(\min_{i=1,2} Q_{\phi_i}(s,\tilde{a}) - \alpha \log \pi_\theta(\tilde{a}|s) \right), \qquad \tilde{a} \sim \pi_\theta(\cdot|s) \tag{13}$$

Algorithm 1 shows the pseudocode for the version of SAC suitable for discrete action spaces, using either the expected or samples update strategies mentioned.

Algorithm 1 Discrete SAC

Initialization: Policy parameters θ, Q-function parameters ϕ_1, ϕ_2, $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$, empty replay buffer $\mathcal{D}$
repeat
  Observe state s and select action $a \sim \pi_\theta(a|s)$
  Execute action a, observe reward r, next state s' and termination status d
  Store transition (s, a, r, s', d) in $\mathcal{D}$
  if time to update **then**
    for j in range(number of updates) **do**
      Sample random minibatch of transitions B from replay buffer $\mathcal{D}$
      Compute targets y for Q functions using either (5) or (6)
      Update Q-functions by one step gradient descent, for $i = 1, 2$, using (8)
      Update policy by one step of gradient ascent using either (12) or (13)
      Update target networks with (4)
    end for
  end if
until convergence

3. Experiments

We have performed experiments using our own implementation on a SAC agent, following algorithm 1. Instead of running until convergence, agents were run for a total of 10^6 environment steps. After every 1000 environment steps, we evaluated the episode return over the policy network in a fresh environment, greedily selecting the action with the highest probability at each step. Each configuration was run for 5 repetitions, over which the average evaluation returns and confidence intervals were plotted. Both the average evaluation returns and confidence intervals were smoothed using a Savitzky-Golay filter with window size 31. The update frequency, number of updates per update step and the number of neurons in the two hidden neural network layers were parameterized. The following hyperparameters were used as the baseline configuration unless otherwise stated: learning rate $= 0.001$, $\gamma = 0.99$, hidden dim $= 64$, $\alpha = 0.5$, buffer size $= 10^5$, batch size $= 100$, learning starts $= 1000$, $\rho = 0.005$, update freq $= 5$, update num $= 1$. Gradient descent updates for all networks were performed using the Adam optimizer (Kingma & Ba, 2014).

3.1. Environment

All experiments were performed on the CartPole-v1 environment. This environment simulates a classic control problem where a pole is attached to a cart, which moves along a frictionless track. The goal is to balance the pole by applying forces in the left and right direction on the cart. The action space is binary and corresponds to actions for pushing the cart either to the left or to the right. The state space consists of 4 continuous values: the cart position, the cart velocity, the pole angle, and the pole angular velocity. A default reward of $+1$ is given for every step taken, with an episode time limit of 500 steps. An episode is terminated if either the pole angle is greater than $\pm 12°$, the cart position is greater than ± 2.4, or the episode time limit is reached.

3.2. Temperature parameter

For the first experiment, we looked at the impact of the temperature parameter α on the learning performance on a SAC agent using expected updates for both the actor and critic. Before testing, there were significant doubts of its impact. As the return of a CartPole-v1 episode is between 1 and 500, the Q-values on average take on much higher values compared to the entropy terms. This then raises the question how a value of $0 < \alpha \leq 1$ could possibly influence the balance between exploration and exploitation, as the entropy regularized state values are still dominantly determined by the Q-values.
However, as can be seen in Figure 1, these concerns were unnecessary, as even comparably small changes of the value of α can have a significant impact on the learning stability of the algorithm. As is shown, the value of 0.5 reaches a reward of 500 the quickest, closely followed by the α value of 1.0. We can clearly see the effect of the increased measure of exploration at higher values of α, as the learning at early stages appears to escape local optima quicker compared to lower values. Most notable is the result when the entropy term is completely absent; when α has a value of 0.0. In

that case, the algorithm learns much more slowly, and is also very unstable at the later stages of learning. This shows that, contrary to intuition, the addition of entropy makes the SAC algorithm much more stable than when it would be absent, even at small α values.

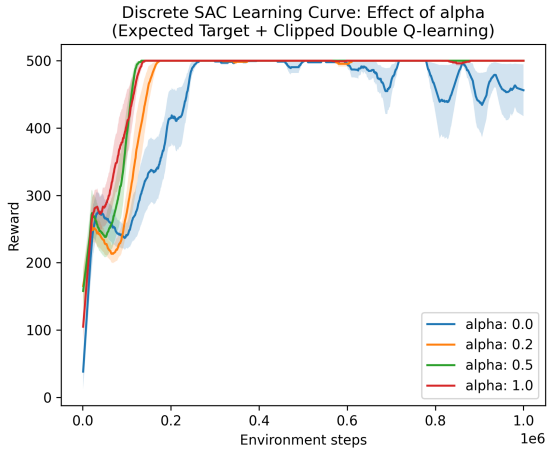

Figure 1. Evaluation learning curve for Discrete SAC on CartPole-v1. Showing effect of the temperature parameter α with Expected Updates and Clipped Double Q-Learning.

3.3. Clipped Double Q-learning

For the second experiment, we again looked at the impact of the temperature parameter α on the learning performance, but now on a SAC agent that does not utilize Clipped Double Q-learning, and instead uses a single Q-network. Figure 2 shows that this change severely impacts the learning stability at all α values, but has a bigger impact on lower values of α. This results signifies that Clipped Double Q-learning has a big stabilizing effect on the learning process, and makes the algorithm less sensitive to non-optimal values of α. This can in practice be very beneficial as the optimal value of α can change based on the action and reward spaces of different environments.

3.4. Expected vs Sample Updates

For the third experiment, we switch from expected updates to sampled updates for both the actor and critic. This experiment has the goal of showing how our SAC agent would perform on a continuous action space, or a very large discrete action space where it is infeasible to perform expected updates. The results in figure 3 show that while the relative performance between the different values of $\alpha > 0$ stay similar compared to figure 1, the overall learning speed and convergence stability of the agents is severely hampered by the sampled updates. This can be explained by the increased variance introduced by the sampling approach, resulting in less precise gradient descent steps that are more likely to

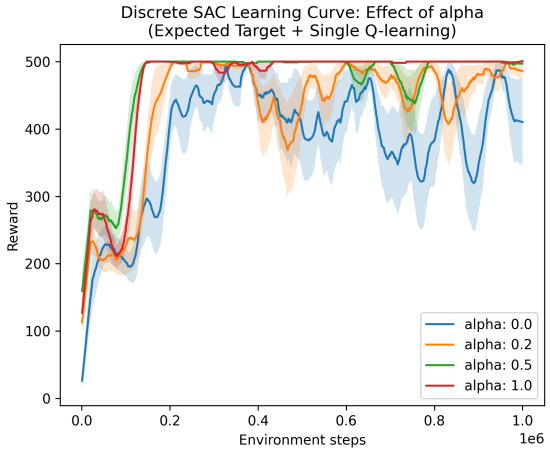

Figure 2. Evaluation learning curve for Discrete SAC on CartPole-v1. Showing effect of the temperature parameter α with Expected Updates and Single Q-Learning.

overshoot the optimal targets. The learning performance using an α value of 0.0 appears to be impacted much more severely, indicating that the entropy term has a much more meaningful contribution to the learning performance here compared to the expected update agents.

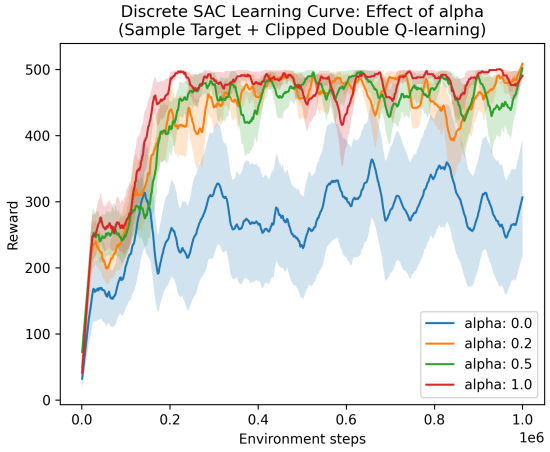

Figure 3. Evaluation learning curve for Discrete SAC on CartPole-v1. Showing effect of the temperature parameter α with Sampled Updates and Clipped Double Q-Learning.

As both the absence of Clipped Double Q-learning and the use of sampled updates are shown to negative impact learning performance, we were interested in whether the negatively contributions of these design choices stack. We therefore performed a final experiment on these design choices where we combined the sampled target updates with a single Q-network. The results in figure 4 show that we indeed see learning curves that can be characterized by a combination of our observations in figures 2 and 3. The lack of Clipped

Double Q-learning again shows more sensitivity to values of α on the learning performance, while the use of sampled updates again impacts the learning speed and convergence stability.

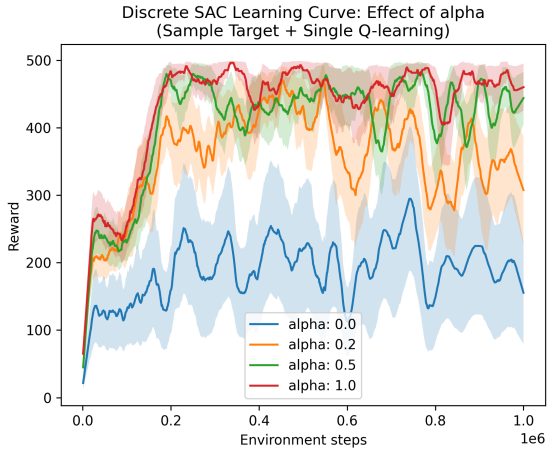

Figure 4. Evaluation learning curve for Discrete SAC on CartPole-v1. Showing effect of the temperature parameter α with Sampled Updates and Single Q-Learning.

3.5. Update frequency

For the final experiment, we looked at the impact of the update frequency, or in other words the number of environment steps taken before one or more updates are performed. This causes the replay buffer to collect more recent environment transitions before each update, and therefore directly influences the learning speed of the algorithm. Compared to on-policy algorithms like REINFORCE and basic Actor-Critic methods, we can reuse old experiences for learning, decoupling the number of environment steps between updates and the number of transitions on which each update is performed. Decreasing the update frequency would then in theory also increase the sample efficiency, and therefore the learning speed of the algorithm. The results in figure 5 confirm this hypothesis. We see that when doubling the update frequency from 5 to 10, the agent also converges around double the number of environment steps. The inverse has a similar effect, seeing that an agent that updates every environment step appears to learn much quicker. We do however see diminishing returns using this smaller update frequency, as the convergence behavior of this agent appears to be less stable, likely due to the decreased amount of exploration caused by the quick learning. It is also worth noting that the runtime of the algorithm linearly increases with the number of updates, which further discourages updating every single environment step.

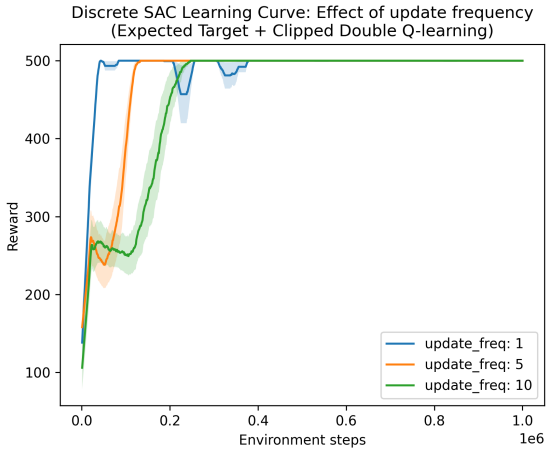

Figure 5. Evaluation learning curve for Discrete SAC on CartPole-v1. Showing effect of the frequency of updates compared to the number of environment steps taken.

4. Discussion

As has been shown in Section 3, the outcomes of the experiments show a clear trend. For starters, Clipped Double Q-Learning vastly outperforms Singular Q-Learning. While it will introduce a larger overhead, as more networks cost more space and time, the SAC agent is clearly more stable when using Clipped Double Q-learning.
It has also been shown that the SAC agent works for both expected and sampled updates, meaning that it would work for both discrete and continuous action spaces. In an environment such as CartPole-v1, with a small discrete action space, expected updates do perform better than sampled updates, but as with larger discrete or even continuous action spaces using expected updates is unfeasible such comparison has no general merit.
As for the parameters that have been tested, knowing we are using expected targets and Clipped Double Q-Learning, the value for α appears to be most optimal at 0.5. For the update frequency, it has been shown that smaller values lean faster but also have diminishing returns, both in terms of stability and increased runtime. Therefore, the update frequency need to be carefully chosen to strike the right balance for the given environment.

Figure 6 shows a comparison between the performance of SAC using our baseline configuration, and some of the algorithms studied during the previous assignments: DQN, REINFORCE and Advantage Actor-Critic (A2C). These algorithms were configured to use the same hyperparameters as the SAC baseline when possible: all use same learning rate and γ, and DQN uses the same buffer size, batch size and update freq. We see that SAC is the best performing algorithm. It shows similar convergence behavior as A2C, but

does so at a much faster learning speed and with less variance between repetitions. This faster learning speed can be attributed to the higher sample efficiency compared to A2C. While SAC is most similar in design to DQN, their performance is the furthest apart. This difference can partially be explained by the extra engineering tricks SAC incorporates which we have shown to increase learning stability: Clipped Double Q-learning and entropy regularization.

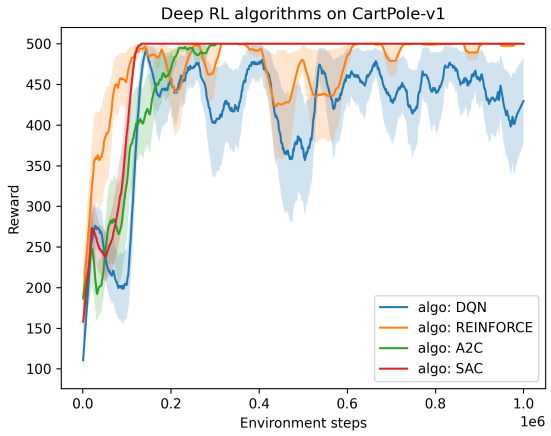

Figure 6. Evaluation learning curves for DQN, REINFORCE, A2C, and SAC on CartPole-v1, all using hyperparameters equal to the baseline SAC configuration when possible.

5. Conclusion

For this assignment, we have studied and implemented the Soft Actor Critic algorithm and tested its performance on the CartPole-v1 environment. We have seen that the addition of entropy regularization has a substantial impact on the learning performance, increasing the convergence stability at later stages in the learning process. We have also shown how Clipped Double Q-learning is used to stabilize the learning process, and how it makes the performance of the algorithm less susceptible to differing hyperparameter settings.

While most values of α resulted in stable convergence of the policy for this environment, that might not be the case for more complex environments. Finding the right value for α could therefore become more challenging. The original authors of SAC have therefore extended the algorithm to include a constrained formulation that automatically tunes the temperature hyperparameter (Haarnoja et al., 2018b). Future work could therefore include implementing this extension to compare our found best performing α value to the automatically tuned one.

In order to adapt SAC to work on a discrete action space, two update strategies were proposed and implemented for both the actor and critic networks: one that uses the expectation over the action probabilities in the current policy, and one that samples the actions from the current policy. We have seen that the expected updates lead to more stable convergence in this environment, as the updates are more precise with less variance. As before mentioned, the original SAC algorithm uses the reparameterization trick to update the policy network, without knowing the full probability distribution over all actions, opposed to the policy gradient method we used. The former is known to have lower variance (Haarnoja et al., 2018a), but we were unable to use it on a discrete action space. Further work could therefore include finding techniques to reduce the variance of the policy gradient method, for example by using baseline subtraction, like we did in the previous assignment.

References

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

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., and Levine, S. Soft actor-critic algorithms and applications. *CoRR*, abs/1812.05905, 2018b. URL http://arxiv.org/abs/1812.05905.

Hasselt, H. Double q-learning. In Lafferty, J., Williams, C., Shawe-Taylor, J., Zemel, R., and Culotta, A. (eds.), *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010. URL https://proceedings.neurips.cc/paper_files/paper/2010/file/091d584fced301b442654dd8c23b3fc9-Paper.pdf.

Jiang, H., Xie, J., and Yang, J. Action candidate driven clipped double q-learning for discrete and continuous action tasks, 2022. URL https://arxiv.org/abs/2203.11526.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

Konda, V. and Tsitsiklis, J. Actor-critic algorithms. In Solla, S., Leen, T., and Müller, K. (eds.), *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999. URL https://proceedings.neurips.cc/paper_files/paper/1999/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 518 (7540):529–533, February 2015. ISSN 00280836. URL http://dx.doi.org/10.1038/nature14236.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In Solla, S., Leen, T., and Müller, K. (eds.), *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999. URL https://proceedings.neurips.cc/paper_files/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

Towers, M., Kwiatkowski, A., Terry, J., Balis, J. U., De Cola, G., Deleu, T., Goulão, M., Kallinteris, A., Krimmel, M., KG, A., et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3–4):229–256, May 1992. ISSN 0885-6125. doi: 10.1007/BF00992696. URL https://doi.org/10.1007/BF00992696.