Learning to Walk Efficiently

Anonymous author

Abstract

While the bipedal robot successfully begins to jog after converging in just over 170 episodes, achieving an average score of 300 or more, it struggles significantly in the hardcore environment, often getting stuck on obstacles and failing to achieve similar performance. These results highlight the challenges of balancing exploration and exploitation in complex scenarios and underscore the need for further improvements to handle highly variable and unpredictable environments effectively.

1 Methodology

This paper proposes training a bipedal robot to learn to walk by sampling actions a from a stochastic policy $a \sim \pi(s, \theta)$. We utilise the Soft Actor-Critic (SAC) algorithm [4], where the policy is represented as a Gaussian distribution where actions are sampled based on the mean and standard deviation produced by the policy network. The actions are then squashed using the `tanh` function to ensure they lie within the action space bounds of $[-1, +1]$. These actions correspond to the bipedal walker agent action, as illustrated in Table 1 [2]:

Table 1: Action space of BipedalWalkerV3

	Name	**Min**	**Max**
0	Hip 1 (Torque/velocity)	-1	$+1$
1	Knee 1 (Torque/velocity)	-1	$+1$
2	Hip 2 (Torque/velocity)	-1	$+1$
3	Knee 2 (Torque/velocity)	-1	$+1$

1.1 Architecture

Soft Actor-Critic, SAC, involves three main componenrs: a Value Network, Policy Network (Actor) and a Q-Network (Critic).

1.1.1 Value Network

The Value Network comrises of a multilayer perceptron with three linear layers, each followed by a ReLU activation function. It takes the state as input and outputs a single scalar value representing the state value. We aim the minimise the value loss, given by the equation:

$$L_V = \mathbb{E}_{s \sim \mathcal{D}} \left[\left(V_\psi(s) - \mathbb{E}_{a \sim \pi_\phi(a|s)} \left[Q_\theta(s, a) - \alpha \log \pi_\phi(a|s) \right] \right)^2 \right]$$

[4]

1.1.2 Q-Network (Critic)

The Q-Network $Q_\theta(s, a)$ estimates the value of taking action a in state s. SAC uses two Q-networks to mitigate positive bias:

$$L_Q = \mathbb{E}_{(s,a) \sim \mathcal{D}} \left[\frac{1}{2} \left(Q_\theta(s, a) - y \right)^2 \right]$$

where the target value y is given by:

$$y = r + \gamma \left(\min_{i=1,2} Q_{\theta_i'}(s', a') - \alpha \log \pi_\phi(a'|s') \right)$$

and $\pi_\phi(a'|s')$ is the policy network providing the next action a'.

To minimize the Q-function loss, we perform gradient descent on each Q-network's loss functions

1.1.3 Policy Network (Actor)

The Policy Network $\pi_\phi(a|s)$ generates actions to maximise the reward while incorporating an entropy term to encourage more exploration for newer actions. The policy loss is:

$$L_\pi = \mathbb{E}_{s \sim \mathcal{D}} \left[\alpha \log \pi_\phi(a|s) - Q_\theta(s, a) \Big|_{a=\pi_\phi(s)} \right]$$

.

1.1.4 SAC Algorithm

The SAC algorithm involves the selection of actions and updating parameters. The policy itself is modeled as a Gaussian distribution with the mean $\mu_\phi(s)$ and standard deviation $\sigma_\phi(s)$ parameters outputted by the neural network. Actions are sampled from this distribution and then squashed through a tanh function to ensure they remain incide the action space. This is demonstrated by the equation:

$$a \sim \pi_\phi(a|s) = \tanh(\mu_\phi(s) + \sigma_\phi(s) \cdot \epsilon),$$

where $\epsilon \sim \mathcal{N}(0, 1)$ denotes the Gaussian noise that introduces stochasticity into the action selection process, allowing for exploration.

The parameters of the Policy Network, Q-Networks, and Value Network are updated using the experiences stored in a replay buffer.

To improve sample efficiency, we utilise Emphasizing Recent Experience, as proposed by Wang and Ross [8]. We implement an evaluation window to focus on the most recent transitions in the replay buffer. During training, the replay buffer adapts the sampling strategy to focus on recent experiences. The probability of sampling a transition is adjusted based on how recent it is within the evaluation window. Specifically, the buffer keeps track of recent rewards and calculates the variability to dynamically adjust the emphasis on recent experiences. This strategy allows the SAC algorithm to focus more on recent transitions, allowing more relevant updates to the current policy.

2 Convergence Results

Figure 1 shows the agent's reward progression across episodes. The graph illustrates how the agent's performance improves over time as it gains more experience and refines its policy for the normal environment.

The Vanilla SAC reaches a score of 300 or above many episodes after the SAC-ERE, which first reaches a score of 309 in episode 170. Note: we use *reward* and *score* interchangeably.

After reaching 170 episodes, the agent demonstrates convergence for a period of time. We define convergence as obtaining an average reward of 300 over 100 consecutive trials (episodes) [2].

However, both agents struggle to maintain this convergence, exhibiting many troughs in score across the episodes, albeit SAC-ERE exhibits fewer troughs, demonstrating its superior stability than its vanilla counterpart through emphasising recent experience.

Figure 2 illustrates the agents' score across episodes in the hardcore environment. Contrary to the results shown in Figure 1, the graphs for the hardcore environment show significant

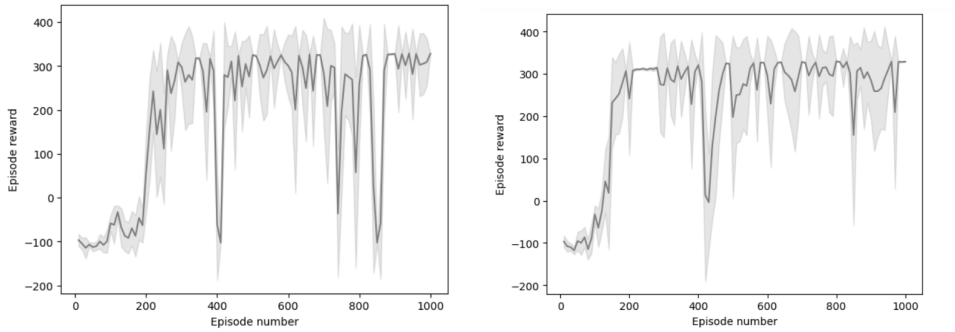

Figure 1: Results for Vanilla SAC (left) and SAC-ERE (right) on normal environment

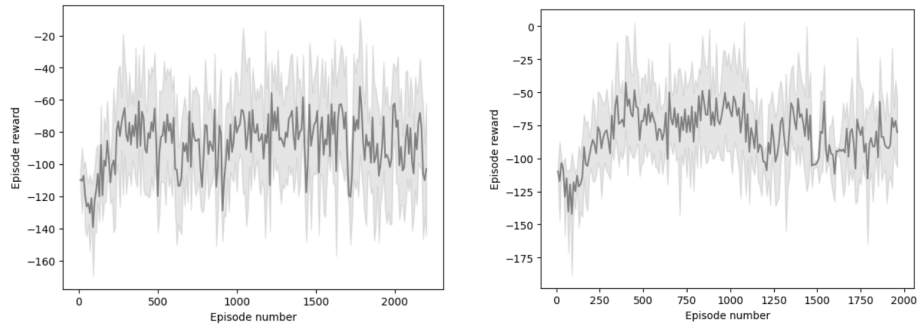

Figure 2: Results for Vanilla SAC (left) and SAC-ERE (right) on hardcore environment

stochasticity. This increased variability in the environment causes both agents to experience fluctiations in their performance, almost never getting a positive reward.

Despite the challenging conditions, SAC-ERE still demonstrates relatively more stability compared to the Vanilla SAC, although both models exhibit numerous troughs and peaks. SAC-ERE, is able to get a higher top score than its vanilla counterpart.

The lack of performance appears to be primarily due to the trade-off between exploration and exploitation. In the hardcore environment, the agents often get stuck on obstacles and struggle to adapt their actions effectively. This may suggest that the agents may over-exploit their current knowledge, leading to sub-optimal behavior such as repeatedly failing to lift their legs over obstacles. The agents fail to explore alternative strategies that might help, and consequently, achieve low scores for the environment.

3 Limitations

In the case of the hardcore environment, the agents fail to get a score of 300 of more. Figure 3 illustrates the agent unable to move past a small obstacle.

Training on the hardcore environment takes a significant amount of time. Often, it becomes apparent only after substantial training that the agent is not performing well. This slow feedback loop can be extremely time intensive, making it very inefficient to improve the model accordingly. SAC algorithm is shown to be extremely sensitive to hyperparameters. Fine-tuning these hyperparameters can be challenging and time-consuming, and suboptimal settings in the experimentation can lead to poor performance.

There is also the issue of exploitation vs exploration, as this seems to be the main culprit as to why the agent performs so poorly on the hardcore environmet. It is possible that the agent is unable to explore alternative strategies, e.g., lifting its front leg ever so often.

Figure 3: Agent stuck on an obstacle in the hardcore environment

Consequently, it fails to traverse through small obstacles. This could potentially be enhanced by the addition of *EmphasizingRecentExperience* to the SAC framework.

Future Work

We could perhaps explore more into the exploration vs exploitation dilemma.A possile implementation could involve using confidence bounds [1].

In addition to the ERE framework, *PrioritizedExperienceReplay* [7] could be integrated to the buffer. This hybrid approach could possibly ensure that important past experiences are not forgotten while still prioritising recent, relevant experiences.

Adaptive exploration frameworks [6] may also benefit the agent by adjusting the exploration rate based on the agent's performance and the environment's complexity.

An exploration of alternative algorithms may also be deemed useful. Algorithms such as *AdaptivelyCalibratedCritics* [3] and *TruncatedQuantileCritics* [5], which are two of the current state-of-the-art methods in this domain. These could potentially address the limitations we observed with out agents.

References

[1] Peter Auer. "Using confidence bounds for exploitation-exploration trade-offs". In: *Journal of Machine Learning Research* 3.Nov (2002), pp. 397–422.

[2] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: `arXiv:1606.01540`.

[3] Nicolai Dorka et al. "Adaptively Calibrated Critic Estimates for Deep Reinforcement Learning". In: *IEEE Robotics and Automation Letters* 8.2 (2023), pp. 624–631. DOI: `10.1109/LRA.2022.3229236`.

[4] Tuomas Haarnoja et al. *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. 2018. arXiv: `1801.01290 [cs.LG]`.

[5] Arsenii Kuznetsov et al. *Controlling Overestimation Bias with Truncated Mixture of Continuous Distributional Quantile Critics*. 2020. arXiv: `2005.04269 [cs.LG]`.

[6] Thomas J Loredo. "Bayesian adaptive exploration". In: *AIP Conference Proceedings*. Vol. 707. 1. American Institute of Physics. 2004, pp. 330–346.

[7] Tom Schaul et al. *Prioritized Experience Replay*. 2016. arXiv: `1511.05952 [cs.LG]`.

[8] Che Wang and Keith Ross. *Boosting Soft Actor-Critic: Emphasizing Recent Experience without Forgetting the Past*. 2019. arXiv: `1906.04009 [cs.LG]`.