# Comparing RL Algorithms on a Bipedal Walker

**Rob E., Visahan S., Omar R., Larisa F., Jithin C. and Ronan S.**
Department of Computer Science
University of Bath
Bath, BA2 7AY
`{rje41, vs929, oir20, lpf31, jc4260, rs3336}@bath.ac.uk`

## 1 Problem Definition

In this project, our objective was to train a bipedal walker to walk across an uneven terrain as efficiently as possible. We used the `BipedalWalker-v3` environment from Gymnasium's Box2D suite [1]. This environment presents a continuous control challenge involving a planar robot with two legs and four joints.

### 1.1 State Space and Action Space

The continuous state space is represented by a 24-dimensional vector that includes:

- The hull's angle, its angular velocity, and its horizontal and vertical speeds.
- The positions and angular speeds for each of the four joints.
- Binary flags of whether each foot is in contact with the ground.
- Ten lidar sensor measurements that help the robot "see" upcoming obstacles.

The learning process is complicated by the fact that the agent does not observe its absolute position. Instead, it must learn to act using relative, local information such as terrain features, encouraging general walking strategies based on sensor feedback. The range of each dimension in the state space is summarised in Table 2 of Appendix A (Section 7.1).

The action space is continuous and is represented by a four-dimensional vector, where each element specifies the motor speed of one of the four joints. Each value is in the range $[-1, 1]$.

### 1.2 Transition Dynamics, Initial Conditions, and Episode Termination

Transitions are deterministic and controlled by the Box2D engine. At each time step, the agent selects an action and the environment simulates the resulting physical interactions. These interactions determine the next state. Small changes in action values can lead to highly non-linear and discontinuous changes in future states. The transition dynamics are unstable and multimodal, making the environment challenging for learning robust control policies.

An episode begins with the walker standing upright at the left end of the terrain, with the hull horizontal and its knees slightly bent. The episode ends when the walker falls, reaches the far end of the terrain, or when the maximum number of time steps (1600) is reached. A time step refers to a single interaction between the agent and the environment.

### 1.3 Reward Function

The agent receives:

- A positive reward for moving forward, proportional to the distance travelled.

- A small penalty for applying a motor torque, discouraging unnecessary and inefficient movements.
- A penalty of -100 if the robot falls and its hull touches the ground.

To "solve" the environment, an agent must achieve a total reward of over 300 within 1600 time steps.

### 1.4 Difficulty of the Problem and Justification for Function Approximation

This task is particularly challenging due to the high dimensionality and continuous nature of both the state and action spaces. Representing all possible states using a fixed grid of values is infeasible; for example, assigning just 10 possible values to each of the 24 state variables results in $10^{24}$ unique state combinations. This scale far exceeds what tabular reinforcement learning methods can handle. The state-action space cannot be exhaustively represented without incurring the curse of dimensionality.

Moreover, the agent must solve a complex credit assignment problem: determining which past actions contributed to success or failure is difficult due to the way outcomes depend on long sequences of actions and non-linear transitions. Tabular methods cannot adequately generalise or learn efficiently in this environment. To address these challenges, function-approximation methods are necessary to solve the problem.

## 2 Background

### 2.1 Solving Locomotion in Continuous Action Spaces

Robotic locomotion is the task of teaching an agent to move efficiently in a given environment. Reinforcement learning has proven to be an effective method for solving this problem. Early efforts in this field concentrated on discrete control tasks, but real-world robotic applications usually work in continuous action spaces, where each motor or joint must be precisely controlled over a continuous range of values [2] [3].

As continuous action spaces in locomotion challenges are high-dimensional by nature, it is difficult for RL algorithms to directly develop efficient control techniques [4]. Given the abundance of real-valued inputs and outputs, a tabular approach cannot scale. Function approximators, such as neural networks, are required to generalise across infinitely many state–action pairs [5]. Sparse rewards and complex reward landscapes further complicate locomotion tasks by trapping agents in local optima, thus limiting efficient exploration [6]. To address these challenges, prior studies have explored techniques like environment design, which can make reward signals more informative and hence training more stable [7], as well as implementing novelty search into the algorithm, which shows that the agent can overcome exploration limitations in high dimensional spaces [6].

Given the characteristics of `BipedalWalker-v3`, we evaluated four Deep RL techniques that are well-suited to continuous control tasks: Deep Deterministic Policy Gradient (DDPG), Twin Delayed DDPG (TD3), Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC).

### 2.2 Deep Deterministic Policy Gradient

DDPG is an off-policy, actor-critic algorithm that is designed for continuous action spaces. It incorporates ideas from Deep Q-Networks (DQN) and deterministic policy gradients, using an off-policy critic network to estimate action values and a deterministic policy for the actor. To improve training stability, it uses experience replay and target networks [8]. While DDPG has demonstrated effectiveness in some control tasks, studies have shown that in the `BipedalWalker-v3` environment it often converges prematurely to suboptimal policies due to its sensitivity to hyperparameters [9].

### 2.3 Twin Delayed DDPG

TD3 addresses DDPG's tendency to overestimate Q-values through clipped double Q-learning, where two independent critic networks are employed and the minimum of both predicted Q-values is used in the target update. It also delays policy updates to allow the critic to learn more accurately, and adds noise to target actions to reduce overestimation and encourage smoother Q-value estimates [10]. A study comparing TD3 and DDPG on the `Walker2D` environment [1], which is similar to `BipedalWalker-v3`, found that TD3 outperformed DDPG in terms of stability, convergence speed, and locomotion efficiency [11].

### 2.4 Proximal Policy Optimization

PPO is an on-policy, policy-gradient method wherein the agent collects data by experimenting in the environment, and then uses that data to update its policy by maximising a "surrogate" objective function using stochastic gradient ascent [12]. We assumed that off-policy methods would perform well on `BipedalWalker-v3`, but included PPO due to its reputation for stability in complex environments. Despite being on-policy, PPO's clipped surrogate objective often leads to more stable performance, making it a strong candidate for comparison. Although PPO is usually less sample efficient than off-policy methods, it is valued for its simplicity and strong performance. A study found that, when optimised using Bayesian techniques, PPO achieved a mean reward of 181.3 over 10 trials in the `BipedalWalker-v3` environment [13]. This performance exceeded that of Advantage Actor-Critic, but remained lower than SAC and TD3 under the same conditions.

### 2.5 Soft Actor-Critic

SAC is an off-policy, Deep RL algorithm based on the max entropy reinforcement learning framework [14]. It optimises a stochastic policy with an entropy bonus, encouraging exploration by favouring policies with higher entropy while simultaneously aiming for high rewards.This objective improves exploration and often leads to more stable and sample-efficient learning, while reducing sensitivity to certain hyperparameters. For `BipedalWalker-v3`, SAC has been shown to achieve competitive performance, with some implementations showing mean rewards of around 300, indicating near-optimal policies [15].

Each of these algorithms offers a promising solution to the challenges posed by environments with continuous action spaces. In this project, we applied and assessed them to better understand their practical performance and to determine which algorithm delivered the most consistent and rewarding outcomes in the `BipedalWalker-v3` environment.

## 3 Method

We split our team of six into three pairs, each focusing on implementing a different Deep RL algorithm: PPO, SAC and TD3. Dividing the work allowed us to develop agents in parallel and explore the strengths and weaknesses of each algorithm in greater depth. To ensure a fair comparison, we kept certain parameters consistent across all implementations, such as the number of episodes (500), and replay buffer size (200,000). However, other hyperparameters were tuned individually for each algorithm to maximise their respective performance. Rather than constraining all agents with uniform settings, this approach allowed us to assess each method's full potential. A summary of the hyperparameters used for each algorithm can be found in Table 3 in Appendix C (Section 7.4).

### 3.1 DDPG Implementation

To establish a consistent baseline amongst all the pairs, we began by implementing the DDPG algorithm. We chose to start with DDPG because it introduces techniques that are shared with more advanced algorithms, so it served as a stepping stone for later development.

In our implementation, the actor network outputs continuous actions, while the critic estimates the Q-value of those actions for a given state. We used experience replay to store past interactions and sample random batches during training, which helps break correlations between experiences and improve stability. We also use soft updates to slowly adjust target networks for more stable learning. To encourage more exploration, we added Gaussian noise to the actions outputted by the actor network before they were applied in the environment. This helps the agent explore in a more natural and consistent way, instead of random jumps in learning. Implementing DDPG first gave us valuable insight into the challenges of training agents in this environment and provided a benchmark against which to compare the improvements of PPO, SAC, and TD3.

### 3.2 TD3 Implementation

We implemented TD3 based on the original design by Fujimoto et al. [10], with hyperparameters fine-tuned for `BipedalWalker-v3`. We matched their critic learning rate of $1 \times 10^{-3}$, but found that reducing the actor learning rate to $1 \times 10^{-4}$ improved performance. Since the actor depends on the critic's accuracy, updating the actor too quickly on an inaccurate critic may amplify errors. A lower actor learning rate allows the critic to adapt faster and provide better supervision. We used standard values of 0.99 for the discount factor and 2 for policy delay (ie. the actor is updated once every two critic updates). For the soft update rate ($\beta$), we chose 0.05, enabling target networks to adapt quickly to changes in the critic and actor. Applying a higher than standard Gaussian noise ($\sigma$) of 0.3 to the actor's output encouraged broader action space exploration, and then decaying the noise by a rate of 0.99 correlated with an improved agent performance. We used a replay buffer size of 200,000 for consistency with our SAC implementation. The actor and critic networks used 256-unit hidden layers, a relatively shallow architecture chosen to reduce the risk of vanishing gradients and over-parameterisation, both of which can destabilise training.

### 3.3 PPO Implementation

We implemented PPO, an on-policy algorithm that updates the policy in small steps to avoid destabilising changes. The actor network outputs the mean of a Gaussian distribution, and the standard deviation is learned as a separate parameter. Actions are sampled from this distribution for better exploration. The critic estimates state values, which are used in Generalized Advantage Estimation (GAE) to reduce variance when computing advantages. Advantage can be defined as a measure of how much better off the agent is by taking a particular action in a given state. GAE adds to this by looking at rewards several steps into the future. We set the discount factor ($\gamma$) to 0.99 for discounting future rewards, and lambda to 0.95 for GAE. This strikes a balance between bias and variance. We trained on batches of 4096 time-steps, running 10 epochs per batch with a mini-batch size of 64. A clipping ratio of 0.2 was used to limit how much the policy could change during updates, keeping learning stable. We also added an entropy term to the loss function to promote more exploration.

### 3.4 SAC Implementation

For our SAC implementation, two independent Q-value critics with 256-unit hidden layers were trained simultaneously, and the minimum of their predicted values was used to compute the Bellman target to reduce over-estimation bias. Target networks were updated using Polyak averaging with $\tau = 0.005$. The actor, critics, and temperature parameter $\alpha$ networks were each optimised using the Adam optimiser with a learning rate of $3 \times 10^{-4}$. The discount factor was set to $\gamma = 0.99$, and $\alpha$ was learned during training to match a target entropy of $-4$, corresponding to the four-dimensional action space. Experiences were stored in a replay buffer with a capacity of 200,000 transitions, and training began once the buffer held at least 128 samples. For each environment step, we performed two critic updates followed by one actor and $\alpha$ update using randomly sampled mini-batches of 128 transitions.

## 4 Results

### 4.1 DDPG Results

Figure 4 in Appendix B (Section 7.2) shows the average episodic reward over five DDPG agents, highlighting both raw returns and their 50-episode moving average. Training is slow with minimal change in reward up to episode 100. This is due to DDPG utilisation of soft updates for the target networks resulting in delayed learning as meaningful changes to the target network takes time. From 100 episodes onwards, we observe a steady but relatively slow growth in rewards up until 500 episodes reaching a peak of around +20 reward for the moving-average curve. This slow learning can be attributed to the Q-value overestimation problem providing inaccurate estimates for actions. This leads to the actor being pushed towards suboptimal actions and misdirects exploration towards falsely overestimated actions, hindering our agents ability to discover better policies quickly.

### 4.2 TD3 Results

The average episodic rewards graph for five TD3 agents can be found in Figure 5 in Appendix B (Section 7.2). As seen in the graph, our TD3 agent shows the same delayed learning initially observed with DDPG, likely due to the use of soft target updates. However after 100 episodes, learning accelerates. This improvement is driven by TD3's use of two critic networks, which helps reduce the overestimation bias common in DDPG. The target Q-value is computed using the minimum of the two critics, providing a more conservative estimate and preventing the actor from being pushed toward overestimated values. Additionally, TD3 adds clipped noise to the target actions, preventing the policy from overfitting by ensuring the critic learns whether the value of an action is locally consistent, not just optimal at a single point. Combined with delayed action updates that allow the critic more time to learn accurate value estimates, these improvements lead to faster learning and higher performance, with rewards exceeding +250 by episode 500.

### 4.3 PPO Results

The average episodic rewards graph for five PPO agents can be found in Figure 6 in Appendix B (Section 7.2). Initially, the rewards are quite low, which is expected as the agent starts off with little knowledge and takes random or poorly informed actions. As training progresses, the reward curve trends upward, showing that the agents are learning and improving their policies over time. The graph shows a lot of fluctuation, which is normal for PPO, because of how it balances between policy stability and exploration through the clipped functions. The curve, especially after episode 100, shows that the hyper parameters ($\gamma = 0.99$, clip = 0.2, $\lambda = 0.95$, 10 epochs per update) and network architecture we chose are effective, resulting in consistent high rewards. The drops in the rewards are likely due to exploration or policy updates that temporarily degrade performance. When deterministically evaluating five agents over 100 episodes (each with a different initial seed), the model averages a reward of 227.3 (Table 1). This exceeds the results from past literature [13], and likely represents the upper limit of what can be achieved with PPO on this environment. However, it falls short of the 300-point threshold typically used to define successful performance.

### 4.4 SAC Results

Figure 7 in Appendix B (Section 7.2) shows the average episodic reward over five SAC agents. Initially, the average return remains near -100, reflecting that most episodes end with the robot falling and receiving a termination penalty. In contrast to the training curves of DDPG and PPO (Figures 4 and 6), SAC's curve (Figure 5) has fewer large oscillations in episodic rewards. This improved stability is a key strength of SAC: its twin-critic architecture helps avoid the Q-value overestimation that often leads to erratic learning behaviour in DDPG and PPO. After episode 250, the reward curve begins to flatten as the replay buffer nears full capacity and becomes increasingly dominated by high-reward trajectories. With SAC sampling uniformly, reduced experience diversity weakens the entropy bonus meant to encourage exploration by rewarding stochasticity in the policy. The agent begins to rely more on consistent, high-return behaviours already stored in the buffer, which leads to more deterministic actions, slower improvement, and occasional sharp drops from failures. Around episode 320, raw returns start to consistently exceed the 300-point "solved" threshold, although the moving average peaks at an episodic reward of 280. During evaluation, the policy runs without exploration noise. In this setting, the agent reliably scores above 300 (Table 1), suggesting it has learned to walk effectively. The gap between training and evaluation is due to the stochastic noise added during training to encourage exploration. Given this, further training is unlikely to lead to significant improvements.

## 5 Discussion

As seen in Figure 1, the SAC agent achieves the highest average reward at the end of training, with the moving average peaking at approximately 280 and occasional spikes in the raw reward curve exceeding 300. This suggests that SAC effectively learns a solution to the environment. TD3 follows, showing a steady increase in average episodic reward, peaking around 250. The PPO agent performs well initially but begins to plateau at around episode 200, ending with a moving average reward of approximately 150. All trained agents demonstrated clear learning progress and outperformed both the baseline DDPG and the random policy.

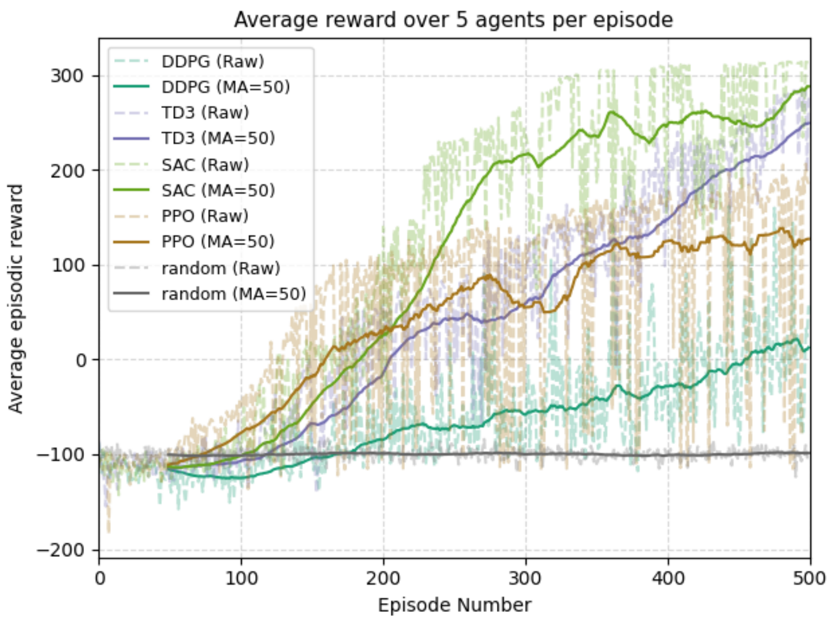


Figure 1: Average episodic reward over 5 agents per algorithm on `BipedalWalker-v3`. The plot compares the training performance of DDPG, TD3, SAC, and PPO against a random action baseline. Both raw episodic rewards and 50-episode moving averages (MA=50) are shown.

Once training was complete, we evaluated the final policy of five randomly seeded agents for each algorithm over 100 episodes. During evaluation, agents acted deterministically without added exploration noise. The results, summarised in Table 1, show that SAC achieved the highest average evaluated return of 312.2, with a remarkably low standard deviation of 10.5, indicating both strong performance and consistency across runs.

Table 1: Mean and standard deviation of returns over 100 deterministic episodes for five agents per algorithm.

| **Agent** | **Average Evaluated Return** | **Standard Deviation** |
|---|---|---|
| Random | −100.3 | 0.9 |
| DDPG | 80.0 | 180.6 |
| PPO | 227.3 | 43.1 |
| TD3 | 262.5 | 17.9 |
| **SAC** | **312.2** | **10.5** |

In contrast, DDPG achieved a much lower average return of 80.0, with a high standard deviation of 180.6, reflecting significant instability. TD3 performed significantly better, with an average return of 262.5 and a lower standard deviation of 17.9, indicating more reliable performance. The improvements introduced in TD3, such as double Q-learning and delayed policy updates, appear to contribute to this increased stability. The high variance observed in DDPG is likely due to some runs benefiting from favourable conditions while others failed to learn effectively. This suggests that DDPG struggles to generalise across different environments, leading to inconsistent performance and a high standard deviation. PPO achieved an average evaluated return of 227.3, with a relatively high standard deviation of 43.1. This variability can be attributed to its on-policy nature, which limits sample reuse and may restrict long-term exploration compared to off-policy methods.

Another notable result is that all off-policy methods required at least 100 episodes to outperform the random agent, with DDPG taking around 150 episodes. This initial delay is likely due to inefficient early exploration and unstable Q-value estimates, which persists until the replay buffer contains a sufficiently diverse set of experiences. In contrast, PPO's on-policy nature and clipped objective function help avoid overly aggressive updates, allowing it to quickly outperform the random baseline. Additionally, since PPO does not rely on a replay buffer, it learns directly from recent experiences, which can lead to faster initial learning. Finally, the average evaluated return for all the trained agents exceeds their training peak, suggesting that adding uncertainty to actions during training (exploration noise), has a significant influence on agent behaviour, even when the noise is gradually decayed. This is a consequence of the environment's non-linear nature.

In conclusion, within the 500-episode training limit, we found that Soft Actor-Critic (SAC) consistently learned successful policies across multiple runs. Owing to its double critic networks and entropy bonus, SAC is both stable and exploratory, enabling it to achieve consistently high rewards. TD3 follows closely, also demonstrating stable performance across runs. To better assess the full learning potential of these methods, future work could relax the episode constraint and evaluate performance over extended training durations for DDPG, TD3, and PPO.

## 6 Future Work

The natural next step for this project would be to build a model capable of solving the more challenging `BipedalWalkerHardcore-v3` environment. This version introduces additional obstacles such as ladders, stumps, and pitfalls to the uneven terrain found in the `BipedalWalker-v3` environment. An image of the `BipedalWalkerHardcore-v3` environment is shown in Figure 8 in Appendix C (Section 7.4). To solve `BipedalWalkerHardcore-v3`, an agent must achieve a reward of 300 within 2000 time steps. We trained an SAC agent on this environment and saved a checkpoint at episode 694, where the training curve appeared to stabilise. The training curve can be seen in Figure 9 in Appendix C (Section 7.4). See Table 7.3 in Appendix C for the hyper arameters used during our experiments on the Hardcore environment. When evaluated over 100 episodes in deterministic mode, this model achieved an average reward of only -6.9. This performance suggests that further improvements to our SAC or TD3 implementations are needed, particularly in stabilising training and improving sample efficiency, which are crucial in the Hardcore environment. One study introduced a forward-looking actor into the TD3 framework, enabling it to solve `BipedalWalkerHardcore-v3` in just a few hours using a single GPU [16]. Additionally, recent work shows that using hint-based models (trained first on the standard environment) can accelerate learning in the Hardcore version [17]. Rainbow DQN is a technique where you combine several improvements to vanilla DQN to improve its stability and performance [18]. Since DDPG's critic update is similar to DQN's, we could also explore how many of these rainbow components can be transferred to DDPG.

## 7 Personal Experience

Throughout this project, we progressed from an understanding of basic tabular RL methods to integrating neural networks to tackle complex problems that traditional tabular approaches could not solve. As a group, we each implemented a state-of-the-art algorithm, gaining strong collective insight into modern RL techniques. As a result, we developed an understanding of the differences between algorithms and how they arrived at distinct solutions. As shown in the accompanying video, each algorithm learned a different solution to the problem. It was rewarding to see the algorithms successfully learn solutions to the problem. However, there were several challenges that needed to be overcome. The first was the large number of hyperparameters that needed tuning. For example, in DDPG, slightly different hyperparameters could mean the difference between an agent that is incapable of learning and one that successfully learns a solution. This involved a long process of trial and error to find "optimal" parameters, despite the possibility that true optimality might never be achieved. In addition to this, the lengthy training times made iterating on the hyperparameters a long process. A solution to this would be to perform a grid search within the parameter space; however, due to the large number of hyperparameters, this may be infeasible. Furthermore, using a neural network to approximate functions added another layer of complexity. The number of hidden units per layer, the overall network depth, and the choice of activation functions also required tuning.

## References

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[2] Amin Abbasi Shahkoo and A. A. Abin. Deep reinforcement learning in continuous action space for autonomous robotic surgery. *International Journal of Computer Assisted Radiology and Surgery*, 18:423–431, 2022.

[3] Xinkai Zuo, Jian Zhou, Fan Yang, Fei Su, Haihong Zhu, and Lin Li. Real-time global action planning for unmanned ground vehicle exploration in three-dimensional spaces. *Expert Syst. Appl.*, 215:119264, 2022.

[4] Yongbin Jin, Xianwei Liu, Yecheng Shao, Hongtao Wang, and Wei Yang. High-speed quadrupedal locomotion by imitation-relaxation reinforcement learning. *Nature Machine Intelligence*, 4:1198–1208, 2022.

[5] J. Santamaría, R. Sutton, and A. Ram. Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive Behavior*, 6:163 – 217, 1997.

[6] Chengyu Hu, Rui Qiao, Wenyin Gong, Xuesong Yan, and Ling Wang. A novelty-search-based evolutionary reinforcement learning algorithm for continuous optimization problems. *Memetic Computing*, 14:451 – 460, 2022.

[7] Daniele Reda, Tianxin Tao, and M. V. D. Panne. Learning to locomote: Understanding how environment design matters for deep reinforcement learning. *Proceedings of the 13th ACM SIGGRAPH Conference on Motion, Interaction and Games*, 2020.

[8] 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.

[9] Jack Dibachi and Jacob Azoulay. Teaching a robot to walk using reinforcement learning, 2021.

[10] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods, 2018.

[11] Xinrui Shen. Comparison of ddpg and td3 algorithms in a walker2d scenario. In *Proceedings of the 2023 International Conference on Data Science, Advanced Algorithm and Intelligent Computing (DAI 2023)*, pages 148–155. Atlantis Press, 2024.

[12] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[13] O. Aydogmus and Musa Yilmaz. Comparative analysis of reinforcement learning algorithms for bipedal robot locomotion. *IEEE Access*, 12:7490–7499, 2024.

[14] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

[15] A. Surriani, O. Wahyunggoro, and A. Cahyadi. A trajectory control for bipedal walking robot using stochastic-based continuous deep reinforcement learning. *Evergreen*, 2023.

[16] Honghao Wei and Lei Ying. FORK: A forward-looking actor for model-free reinforcement learning. *CoRR*, abs/2010.01652, 2020.

[17] Sarod Yatawatta. Reinforcement learning. *Astronomy and Computing*, 48:100833, 05 2024.

[18] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning, 2017.

## Appendices

### 7.1 Appendix A: The Environment

Figure 2 provides a visual reference of the environment. It shows the planar bipedal walker navigating a terrain using its lidar ray.

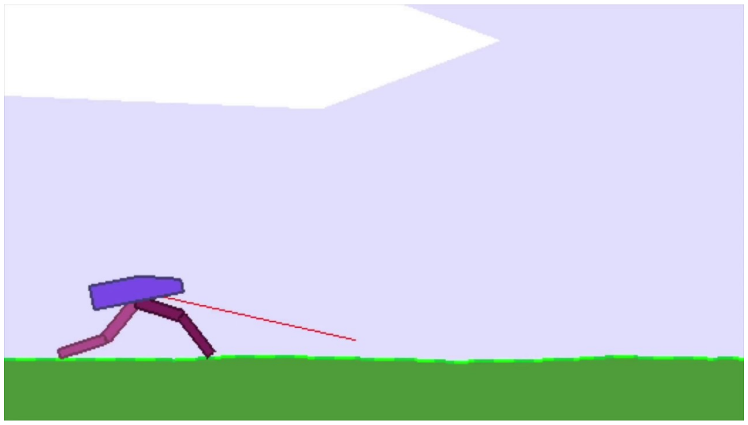


Figure 2: The `BipedalWalker-v3` environment.

Figure 3 shows the bipedal robot receiving a 24-dimensional observation vector and outputting 4 continuous actions to control hip and knee joints for locomotion across uneven terrain.

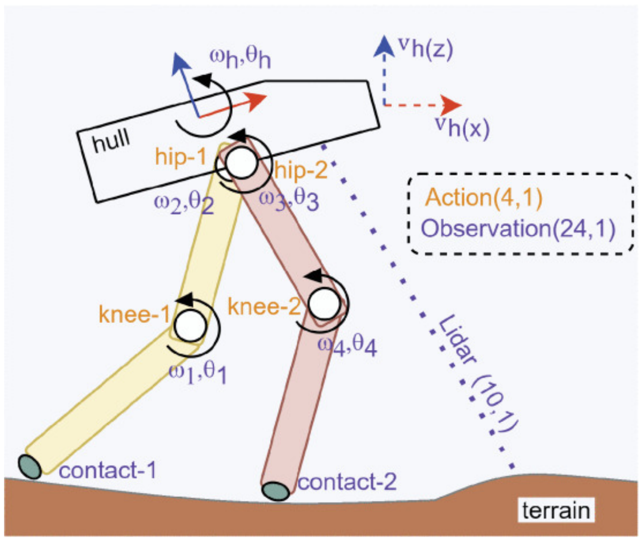


Figure 3: The action and observation states of the bipedal robot [13].

Table 2 summarises the components of the 24-dimensional state space in the `BipedalWalker-v3` environment.

Table 2: State space variables and value ranges in `BipedalWalker-v3`.

| ID | Observation | Interval |
|---|---|---|
| 0 | Hull angle | $[-\pi, \pi]$ |
| 1 | Hull angular speed | $[-5, 5]$ |
| 2 | Hull horizontal speed | $[-5, 5]$ |
| 3 | Hull vertical speed | $[-5, 5]$ |
| 4 | Hip 1 joint angle | $[-\pi, \pi]$ |
| 5 | Hip 1 joint speed | $[-5, 5]$ |
| 6 | Knee 1 joint angle | $[-\pi, \pi]$ |
| 7 | Knee 1 joint speed | $[-5, 5]$ |
| 8 | Leg 1 ground contact flag | $\{0, 1\}$ |
| 9 | Hip 2 joint angle | $[-\pi, \pi]$ |
| 10 | Hip 2 joint speed | $[-5, 5]$ |
| 11 | Knee 2 joint angle | $[-\pi, \pi]$ |
| 12 | Knee 2 joint speed | $[-5, 5]$ |
| 13 | Leg 2 ground contact flag | $\{0, 1\}$ |
| 14–23 | Lidar measurements | $[-1, 1]$ |

## 7.2 Appendix B: Results

Table 3: Hyper parameters used for each algorithm in BipedalWalker-v3.

| Hyper parameter | SAC | TD3 | DDPG | PPO |
|---|---|---|---|---|
| Actor Learning Rate | $3 \times 10^{-4}$ | $1 \times 10^{-4}$ | $1 \times 10^{-4}$ | – |
| Critic Learning Rate | $3 \times 10^{-4}$ | $1 \times 10^{-3}$ | $1 \times 10^{-3}$ | – |
| Alpha Learning Rate | $3 \times 10^{-4}$ | – | – | – |
| Batch Size | 128 | 128 | 128 | 64 |
| Discount Factor ($\gamma$) | 0.99 | 0.99 | 0.99 | 0.99 |
| Soft Update Rate ($\tau$ / $\beta$) | 0.005 | 0.05 | 0.05 | – |
| Entropy Target | $-4$ | – | – | – |
| Replay Buffer Size | 200,000 | 200,000 | 200,000 | – |
| Noise Std. Dev. ($\sigma$) | – | 0.3 | 0.3 | – |
| Policy Delay | 2 | 2 | – | – |
| Exploration Noise Decay Rate | – | 0.99 | 0.99 | – |
| Training Start Delay (in Steps) | – | 1000 | 1000 | – |
| Hidden Dimensions | 256 | 256 | 256 | 64–128 (2 layers) |
| Training Duration | 500 episodes | 500 episodes | 500 episodes | 80,000 timesteps |
| GAE ($\lambda$) | – | – | – | 0.95 |
| Clip Ratio | – | – | – | 0.2 |
| Batch Timesteps | – | – | – | 4096 |
| Epochs per Batch | – | – | – | 10 |

Figures 4, 5, 6, and 7 show the training reward curve averaged over five agents each for DDPG, TD3, PPO, and SAC respectively.

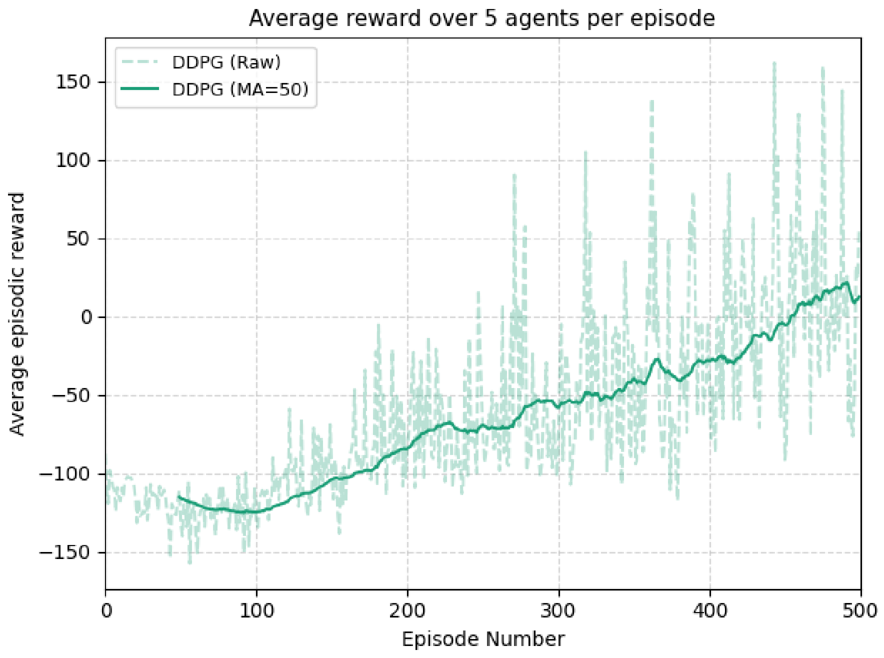


Figure 4: Average training reward of 5 DDPG agents on BipedalWalker-v3.

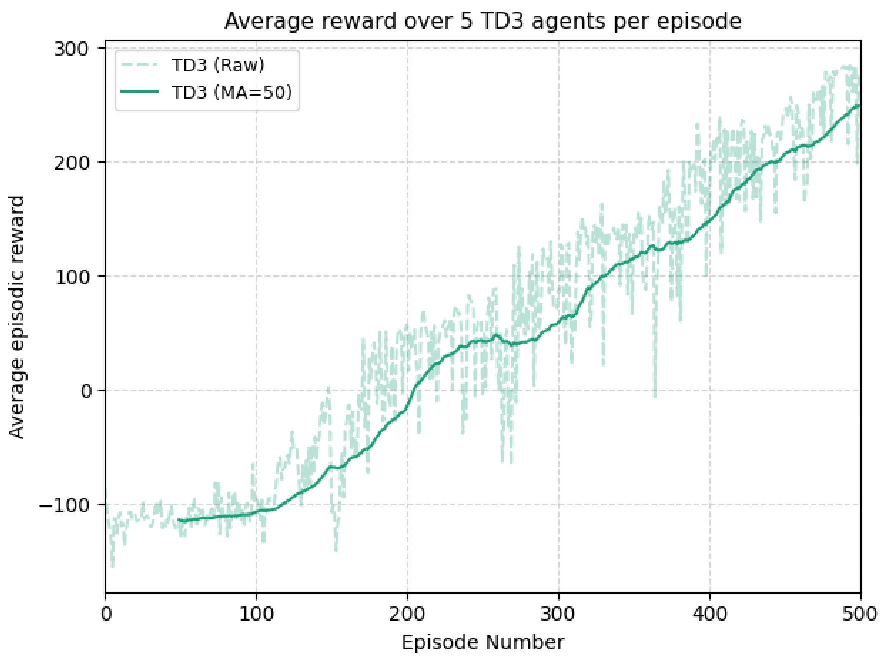


Figure 5: Average training reward of 5 TD3 agents on `BipedalWalker-v3`.

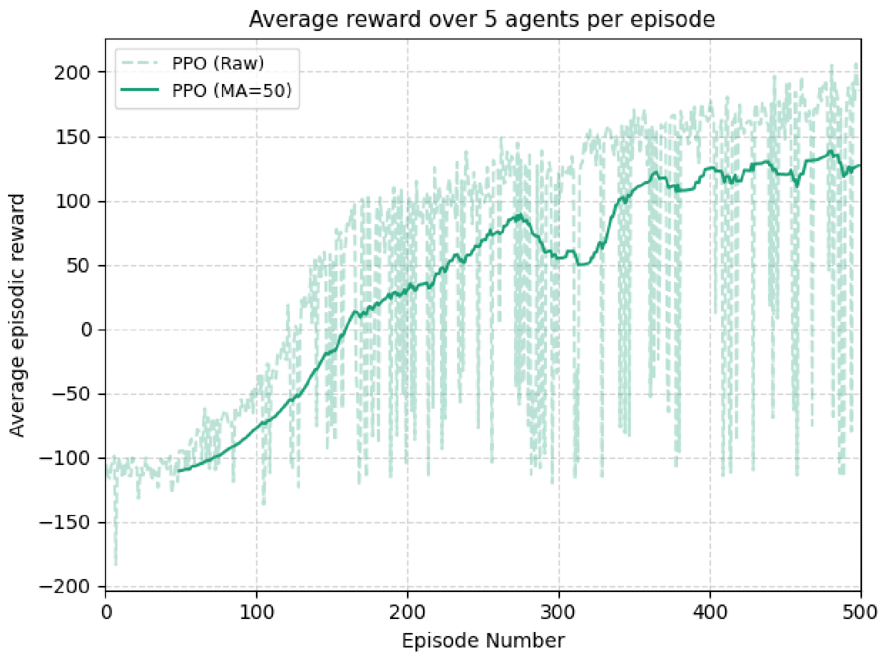


Figure 6: Average training reward of 5 PPO agents on `BipedalWalker-v3`.

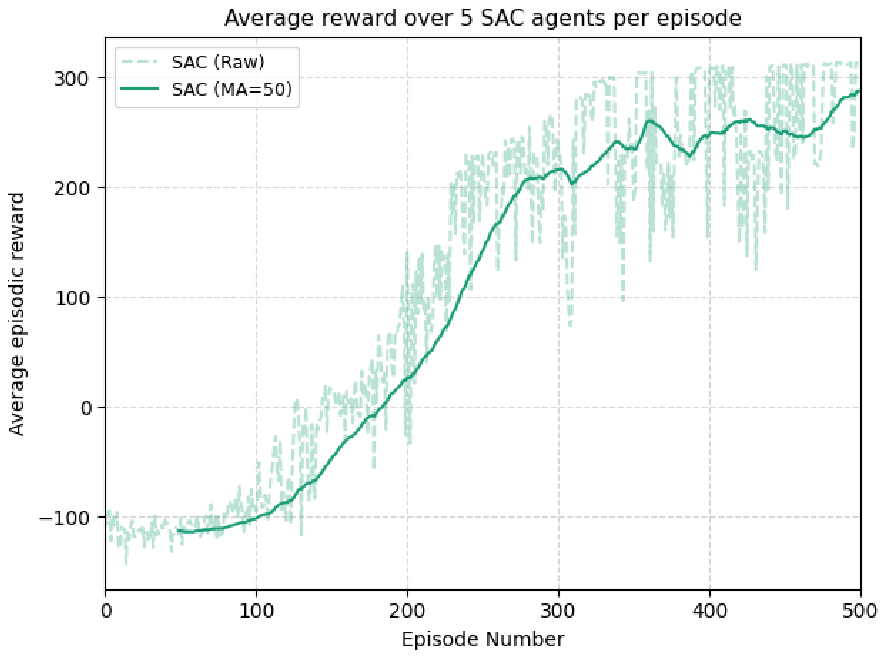


Figure 7: Average training reward of 5 SAC agents on `BipedalWalker-v3`.

## 7.3 Appendix C: Future Work

Figure 8 provides a visual reference of `BipedalWalkerHardcore-v3`, with its additional ladders, stumps, and pitfalls.

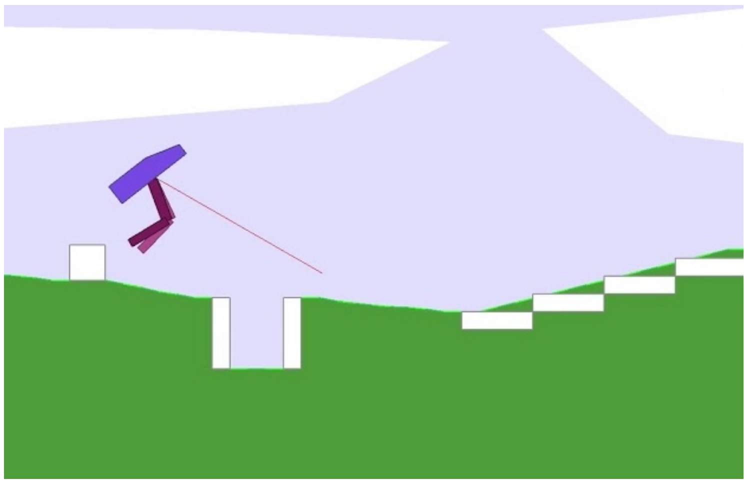


Figure 8: The `BipedalWalkerHardcore-v3` environment.

Figure 9 shows the training reward curve for our SAC model on the `BipedalWalkerHardcore-v3` environment. A checkpoint is saved at episode 694, where the agent appeared to stabilise.

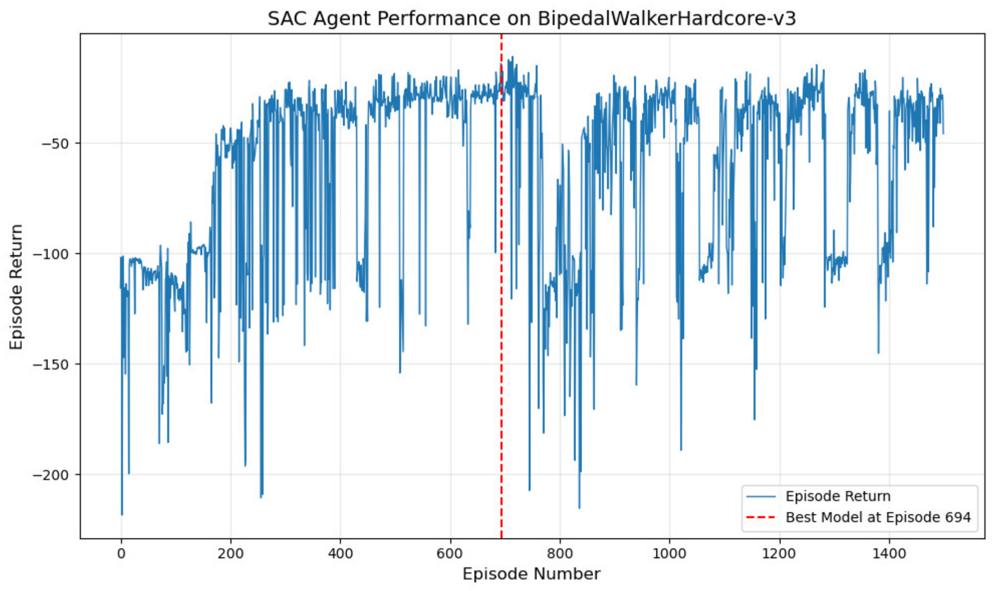
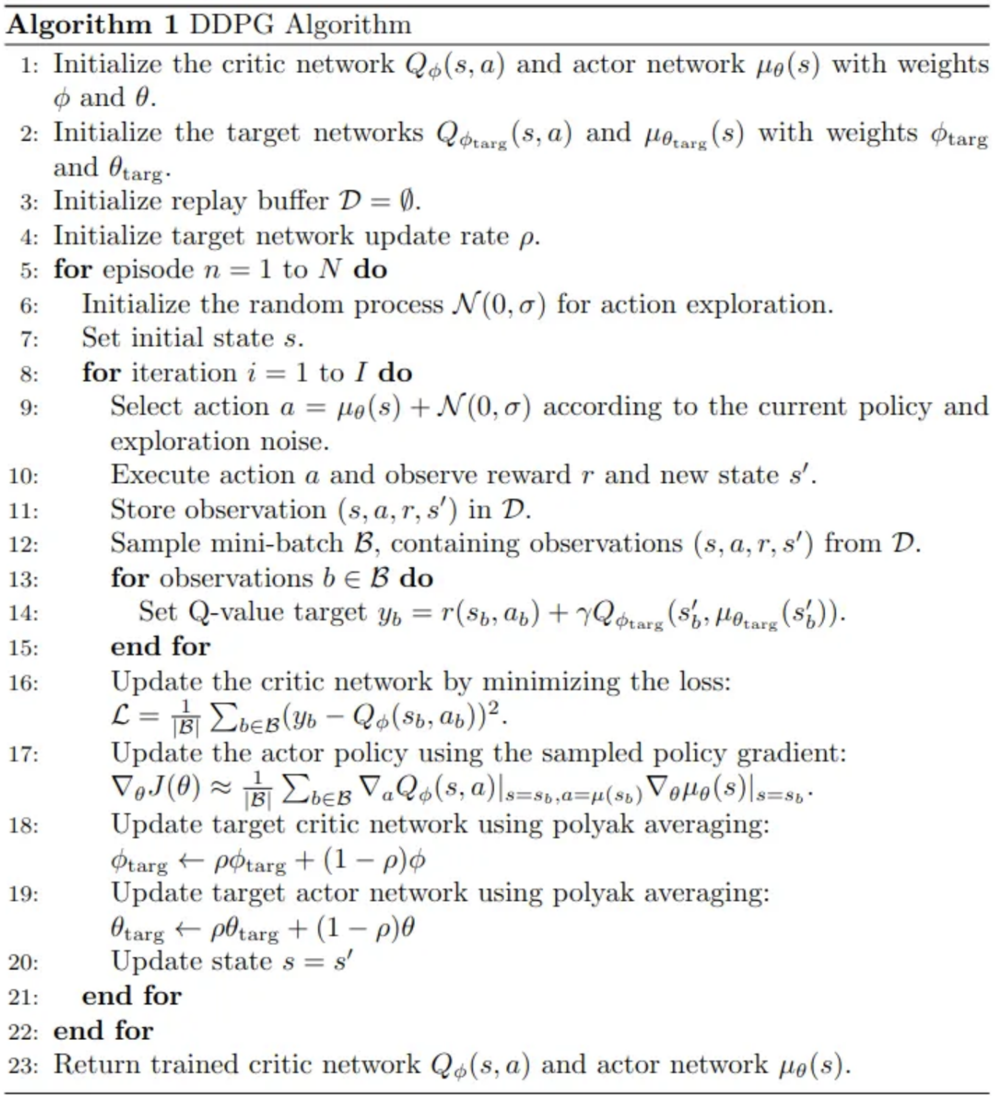


Figure 9: Performace of our SAC Agent on `Bipedal-WalkerHardcore` Environment

Table 4: Hyperparameters used for SAC on `BipedalWalkerHardcore-v3`.

| **Hyperparameter** | **Value** |
|---|---|
| Training Episodes | 1500 |
| Batch Size | 256 |
| Replay Buffer Size | 1,000,000 |
| Actor Learning Rate | $3 \times 10^{-5}$ |
| Critic Learning Rate | $3 \times 10^{-4}$ |
| Alpha Learning Rate | $3 \times 10^{-4}$ |
| Discount Factor ($\gamma$) | 0.99 |
| Soft Update Rate ($\tau$) | 0.005 |
| Target Entropy | $-3$ |
| Hidden Layer Size | 256 |

## 7.4 Appendix D: Pseudo Codes



**Algorithm 1** DDPG Algorithm

```
 1: Initialize the critic network $Q_\phi(s,a)$ and actor network $\mu_\theta(s)$ with weights
    $\phi$ and $\theta$.
 2: Initialize the target networks $Q_{\phi_{\text{targ}}}(s,a)$ and $\mu_{\theta_{\text{targ}}}(s)$ with weights $\phi_{\text{targ}}$
    and $\theta_{\text{targ}}$.
 3: Initialize replay buffer $\mathcal{D} = \emptyset$.
 4: Initialize target network update rate $\rho$.
 5: for episode $n = 1$ to $N$ do
 6:   Initialize the random process $\mathcal{N}(0,\sigma)$ for action exploration.
 7:   Set initial state $s$.
 8:   for iteration $i = 1$ to $I$ do
 9:     Select action $a = \mu_\theta(s) + \mathcal{N}(0,\sigma)$ according to the current policy and
        exploration noise.
10:     Execute action $a$ and observe reward $r$ and new state $s'$.
11:     Store observation $(s,a,r,s')$ in $\mathcal{D}$.
12:     Sample mini-batch $\mathcal{B}$, containing observations $(s,a,r,s')$ from $\mathcal{D}$.
13:     for observations $b \in \mathcal{B}$ do
14:       Set Q-value target $y_b = r(s_b,a_b) + \gamma Q_{\phi_{\text{targ}}}(s'_b, \mu_{\theta_{\text{targ}}}(s'_b))$.
15:     end for
16:     Update the critic network by minimizing the loss:
        $\mathcal{L} = \frac{1}{|\mathcal{B}|}\sum_{b\in\mathcal{B}}(y_b - Q_\phi(s_b,a_b))^2$.
17:     Update the actor policy using the sampled policy gradient:
        $\nabla_\theta J(\theta) \approx \frac{1}{|\mathcal{B}|}\sum_{b\in\mathcal{B}} \nabla_a Q_\phi(s,a)|_{s=s_b,a=\mu(s_b)} \nabla_\theta \mu_\theta(s)|_{s=s_b}$.
18:     Update target critic network using polyak averaging:
        $\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1-\rho)\phi$
19:     Update target actor network using polyak averaging:
        $\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1-\rho)\theta$
20:     Update state $s = s'$
21:   end for
22: end for
23: Return trained critic network $Q_\phi(s,a)$ and actor network $\mu_\theta(s)$.
```

Figure 10: DDPG Pseudo Code

**Algorithm 1** TD3

```
Initialize critic networks $Q_{\theta_1}$, $Q_{\theta_2}$, and actor network $\pi_\phi$
with random parameters $\theta_1$, $\theta_2$, $\phi$
Initialize target networks $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
**for** $t = 1$ **to** $T$ **do**
  Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$,
  $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward $r$ and new state $s'$
  Store transition tuple $(s, a, r, s')$ in $\mathcal{B}$

  Sample mini-batch of $N$ transitions $(s, a, r, s')$ from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
  **if** $t$ mod $d$ **then**
    Update $\phi$ by the deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
    $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
  **end if**
**end for**
```

Figure 11: TD3 Pseudo Code

**Algorithm 2 Proximal Policy Optimization (PPO)**

```
1:  Initialize actor $\mu: S \rightarrow R^{m+1}$ and $\sigma: S \rightarrow diag(\sigma_1, \sigma_2, \dots, \sigma_{m+1})$
2:  for i = 1 to M do
        Run policy $\pi\theta \sim N(\mu(s), \sigma(s))$ for T timesteps and collect $(s_t, a_t, r_t)$
        Estimate advantages $\hat{A}_t = \sum_{t'>t} \gamma^{t'-t} r_{t'} - v(s_t)$
        Update old policy $\pi_{old} \leftarrow \pi_0$
3:      for j = 1 to N do
            Update actor policy by policy gradient:
                $\sum_i \nabla_\theta L_i^{CLIP}(\theta)$
            Update critic by:
                $\nabla L(\phi) = -\sum_{t=1}^{T} \nabla \hat{A}_t^{\,2}$
4:      end for
5:  end for
```

Figure 12: PPO Pseudo Code

**Algorithm 3 Soft Actor-Critic (SAC)**

```
1:  Initialize parameter vectors $\psi, \bar{\psi}, \theta, \phi$
2:  for each iteration do
3:      for each environment step do
            $a_t \sim \pi_\phi(a_t|s_t)$
            $s_{t+1} \sim P(s_{t+1}|s_t, a_t)$
            $D \leftarrow D \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$
4:      end for
5:      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 $i \in \{1,2\}$
            $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
            $\bar{\psi} \leftarrow \tau\psi + (1-\tau)\bar{\psi}$
6:      end for
7:  end for
```

Figure 13: SAC Pseudo Code