Görkem Kadir Solun 22003214 - CS461 Project 3 Report - The Twin Delayed Deep Deterministic Policy Gradient (TD3)

Table of Contents

What is TD3

The Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm is a reinforcement learning method for continuous action spaces. It builds upon the Deep Deterministic Policy Gradient (DDPG) framework by addressing critical issues such as overestimation bias and instability in learning.

TD3 uses two separate critic networks to estimate the value of actions. During updates, the minimum value predicted by these two critics is used, which reduces the overestimation bias inherent in DDPG.

$$y = r + \gamma \min_{i=1,2} Q_{\theta' i}(s', \pi_{\phi'}(s') + \epsilon)$$

The actor policy network is updated less frequently than the critics. This delay ensures that the value estimates used for policy updates are more stable and reliable, reducing the likelihood of propagating errors.

$$\nabla_\phi J(\phi) = E_{s\sim B}[\nabla_a Q_{\theta 1}(s,a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)]$$

TD3 adds slight, clipped noise to the target actions to improve learning stability. This regularization technique prevents the policy from overfitting to narrow peaks in the value function and promotes robustness to small perturbations in the policy.

$$a' = \pi_{\phi'}(s') + clip(\epsilon, -c, c), \quad \epsilon \sim N(0,\sigma)$$

Algorithm Steps

Initialize critic networks $Q_{\theta 1}, Q_{\theta 2}$ and actor π_ϕ with random parameters. Create target networks $\theta_1', \theta_2', \phi'$ initialized to match the primary networks.

For each time step:

Select an action with exploration noise. $a \sim \pi_\phi(s) + \epsilon, \epsilon \sim N(0,\sigma)$.

Store transition (s, a, r, s') in replay buffer B.

Sample a mini-batch from B and compute the target.

$$y = r + \gamma \min_{i=1,2} Q_{\theta' i}(s', \pi_{\phi'}(s') + \epsilon)$$

Update the critics by minimizing the loss.

$$L(\theta_i) = \frac{1}{N} \sum (Q_{\theta i}(s,a) - y)^2$$

Every d-th step, update the actor and target networks.

$$\phi \leftarrow \phi + \alpha_\phi \nabla_\phi J(\phi), \qquad \theta_i' \leftarrow \tau\theta_i + (1-\tau)\theta_i'$$

By leveraging the minimum of two critics, TD3 minimizes systematic errors in value estimation. Delayed updates and target smoothing prevent overfitting and ensure smoother policy improvement. TD3 consistently outperforms DDPG and other algorithms on benchmark tasks in continuous control. TD3 has demonstrated superior performance to DDPG and other state-of-the-art algorithms in continuous control tasks. TD3 achieves more stable and efficient learning in reinforcement learning environments by mitigating overestimation and reducing variance in value estimates.

Implementation

The TD3 algorithm is implemented to solve the Lunar Lander Continuous environment in a reinforcement learning framework. TD3 addresses overestimation bias in action-value function estimation by incorporating multiple innovations, such as maintaining two critics, policy smoothing, and delayed actor updates. This implementation leverages PyTorch for constructing neural networks and Gymnasium for environment interaction.

Replay Buffer

The implementation includes a replay buffer to store past experiences, facilitating off-policy learning. It is a fixed-size data structure that efficiently stores state, action, reward, next state, and signal tuples. The buffer allows the agent to sample mini-batches of experiences uniformly, mitigating temporal correlations and enhancing training stability.

Neural Network Architectures

Actor-Network

Maps the state space to continuous action space using a fully connected architecture. Employs a scaled hyperbolic tangent activation at the output layer to ensure actions remain within valid bounds. Comprises two hidden layers with 400 and 300 neurons, activated by ReLU functions.

Critic Network

Two critic networks are designed to estimate the Q-values (state-action values). Each critic has an independent three-layer structure that combines state and action inputs in the first layer. Outputs a single scalar value for each Q-value estimate.

Target networks for both the actor and critics are introduced to stabilize training by slowly updating their parameters using soft updates.

Core TD3 Innovations

Twin Critics

The algorithm reduces overestimation bias by using two independent critic networks. During training, the smaller of the two Q-value estimates is used to compute the target.

Delayed Policy Updates

The policy network (actor) is updated less frequently than the critic networks. This approach ensures that the critics are better optimized before being used to improve the actor, enhancing stability.

Target Policy Smoothing

Noise is added to the target policy during critic target computation to avoid exploiting sharp local minima in Q-value estimates. This noise is clipped to remain within a feasible action range.

Training Process

Random seeds are set for reproducibility, and the environment is initialized using Gymnasium. The agent explores using a random policy during an initial phase to populate the replay buffer.

Actions are selected by the actor network with added noise for exploration during training episodes. The environment transitions are recorded in the replay buffer.

Mini-batches are sampled from the replay buffer to update the networks. The critics are trained using the Bellman equation, leveraging the smaller Q-value estimate for stability. The actor network is updated using deterministic policy gradients, maximizing Q-values as the critics estimate. Parameters of target networks are softly updated towards their respective online networks after every training step.

Hyperparameter Configuration

Discount factor for future rewards (γ): 0.99.

Learning rates for actors and critics: 10^{-3}.

Noise parameters for exploration and target policy smoothing.

Soft update parameter (τ): 0.005.

Mini-batch size: 100.

Policy update frequency: every two steps.

Evaluation Strategy

Training is conducted using multiple random seeds to assess robustness and generalizability. Rewards are averaged over 100 consecutive episodes to monitor performance stability. The implementation saves model weights for reproducibility and comparison. Training terminates when no significant improvement over a predefined number of evaluations exists.

Termination and Finetuning

Results and Graph Generation

Learning curves showing reward progression across episodes for different seeds are plotted. The mean and standard deviation of rewards across seeds are computed and visualized to illustrate robustness. Data is saved in CSV and NPY formats for detailed analysis and reporting.

The results were produced by training the TD3 agent in the Lunar Lander Continuous environment over multiple episodes and with six distinct random seeds to ensure robustness.

The agent's episodic reward was recorded across each episode to track learning progress. This reward represents the cumulative score obtained in each episode by successfully navigating the Lunar Lander environment.

The training process was repeated for six random seeds to mitigate the effects of outlier scenarios caused by environmental stochasticity or initialization biases.

After training, the rewards from all seeds were averaged for each episode to produce a smooth learning curve. This average provided a robust indicator of performance progression.

The graph plotted the episode rewards over time (x-axis: episodes, y-axis: rewards) for individual seeds and their averaged values. A shaded region representing one standard deviation around the mean reward curve was included to illustrate performance variability between seeds. Results for each seed were visualized individually alongside the means to demonstrate convergence behavior and consistency.

Fine-Tuning of Models

The fine-tuning process involved systematically adjusting hyperparameters and training settings to optimize the agent's performance.

Replay Buffer

The replay buffer was adjusted for sufficient capacity to store past experiences, ensuring diversity in training samples and avoiding catastrophic forgetting. Batch sizes for training were fine-tuned to balance computational efficiency and training stability.

Network Architecture

The actor and critic networks used three-layer architectures with rectified linear activation functions to balance model complexity and computational overhead. Output scaling via tanh activation ensured that actions remained within valid bounds.

The learning rates for the actor and critic optimizers were fine-tuned based on the agent's observed convergence rate. Noise parameters, including the policy noise and noise clipping thresholds, were calibrated to optimize exploration and exploitation dynamics during policy updates. The target smoothing coefficient was adjusted to reduce variance and improve stability in target updates.

The frequency of policy updates was tuned every two iterations to allow the critic to converge adequately before updating the actor, reducing instability due to premature policy updates.

Training sessions employed an early stopping mechanism based on sustained improvement in average rewards. This avoided overfitting and ensured computational efficiency.

Input states were normalized to improve the stability of gradient descent and ensure uniform training dynamics across diverse state distributions.

By iteratively adjusting these elements and analyzing the resultant performance, the TD3 agent demonstrated consistent improvement, eventually achieving the target average reward of over 200 across episodes. This result satisfied the performance threshold specified in the project guidelines.

Results and Evaluation

Each plot illustrates the agent's episodic rewards for different random seeds over time. Additionally, an aggregated graph provides the average performance across seeds, with standard deviations for robustness analysis. The evaluation is based on the reward progression, starting from the initial exploration phase and moving toward convergence.

Best average reward per 100 episodes	Seed 0	Seed 1	Seed 2	Seed 3	Seed 4	Seed 5
100	-218.25	-197.51	-214.04	-196.41	-201.56	-232.49
200	-5.07	-8.07	-58.17	-57.57	-47.34	-91.20
300	32.80	191.07	40.22	10.82	187.25	5.20
400	191.09	210.65	171.09	177.88	237.52	235.29
500	223.50	249.26	210.79	202.15	243.88	245.00
600	237.09	-	232.83	234.47	247.82	252.95
700	238.75	-	266.66	-	259.91	258.29
800	245.67	-	269.5	-	260.43	261.83
900	260.44	-	280.29	-	273.07	266.51
1000	269.97	-	284.80	-	282.58	273.09
1100	271.47	-	293.60	-	-	274.29
1200	278.63	-	-	-	-	276.74
1300	-	-	-	-	-	283.37
1400	-	-	-	-	-	285.23

Table 1 Best average reward per 100 episodes

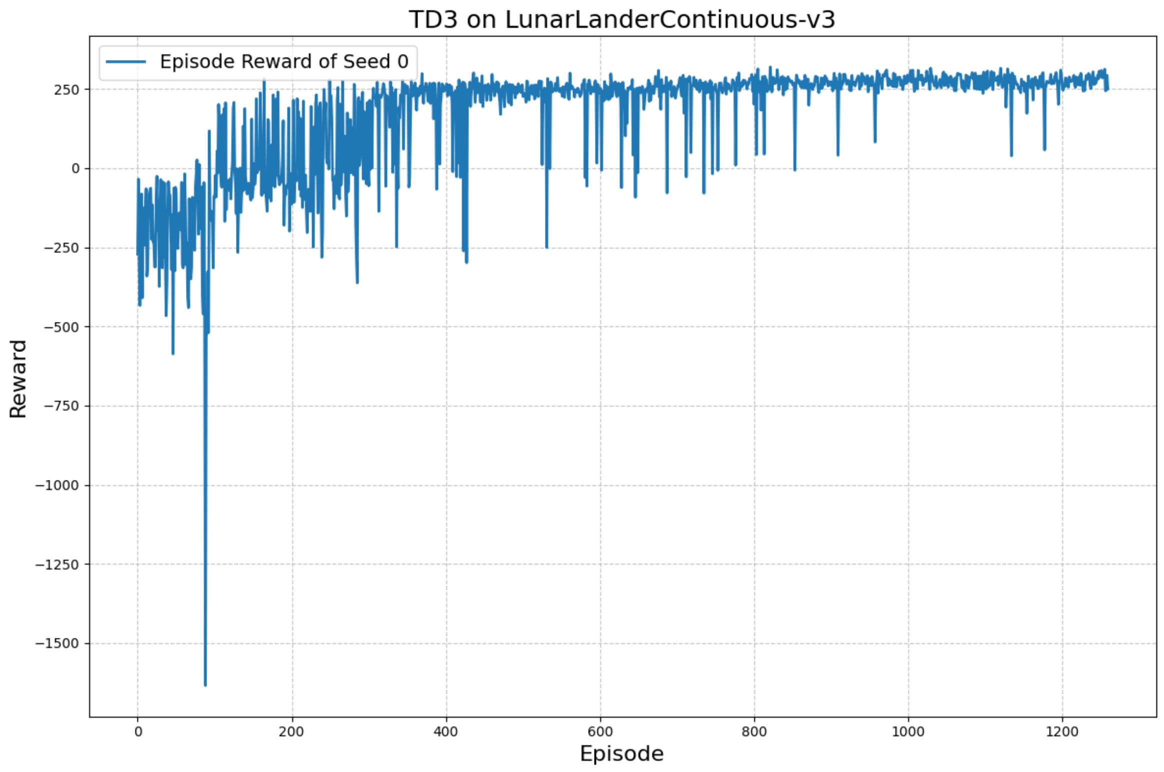

Figure 1: Episode Reward of Seed 0

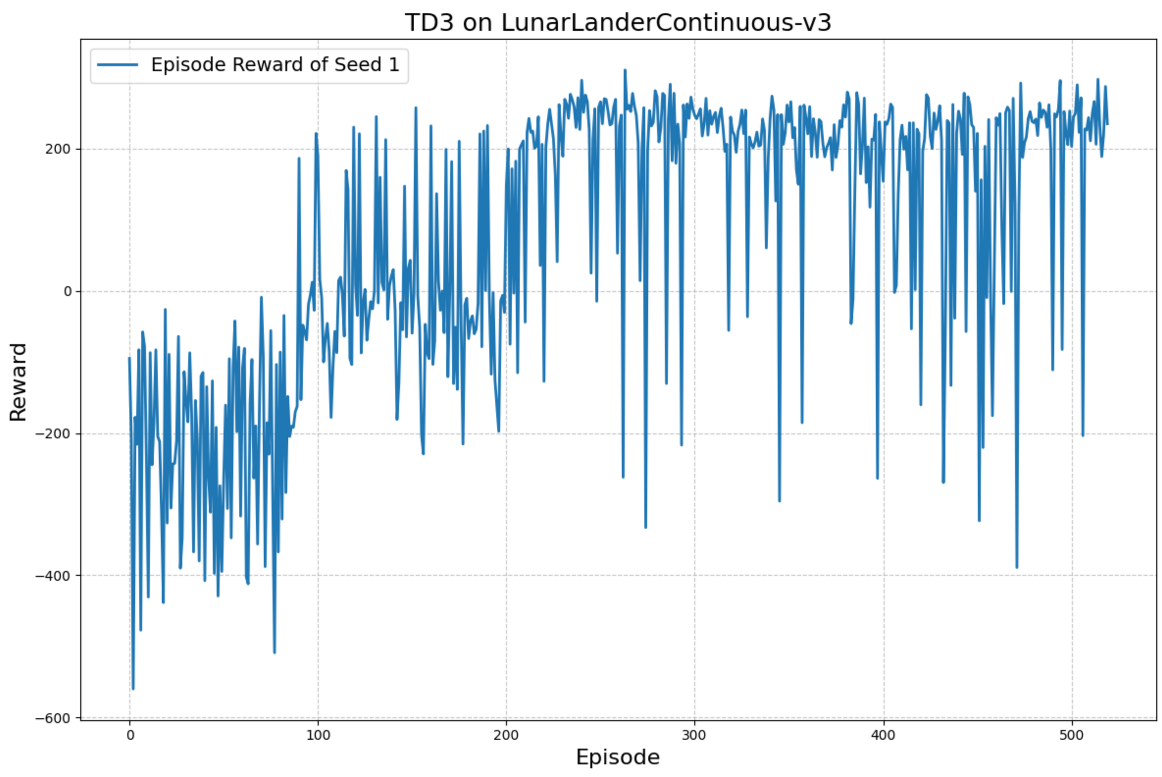

Figure 2 Episode Reward of Seed 1

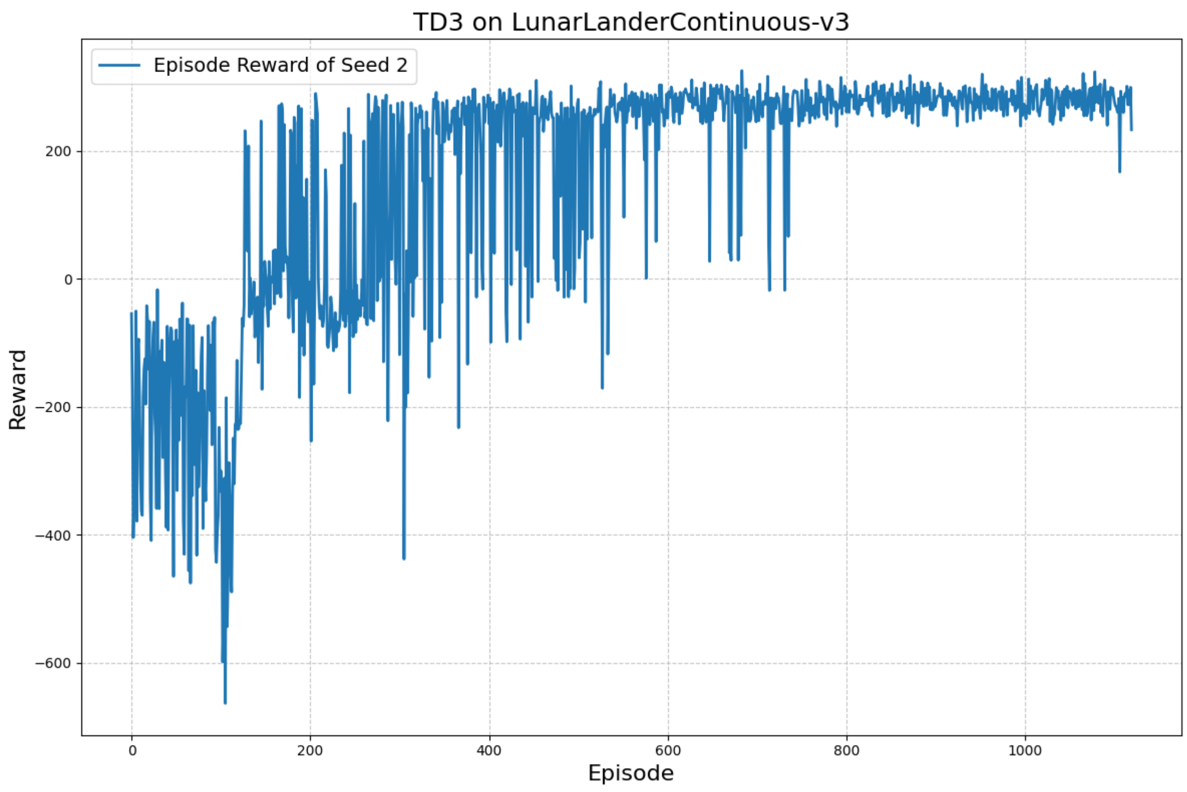

Figure 3 Episode Reward of Seed 2

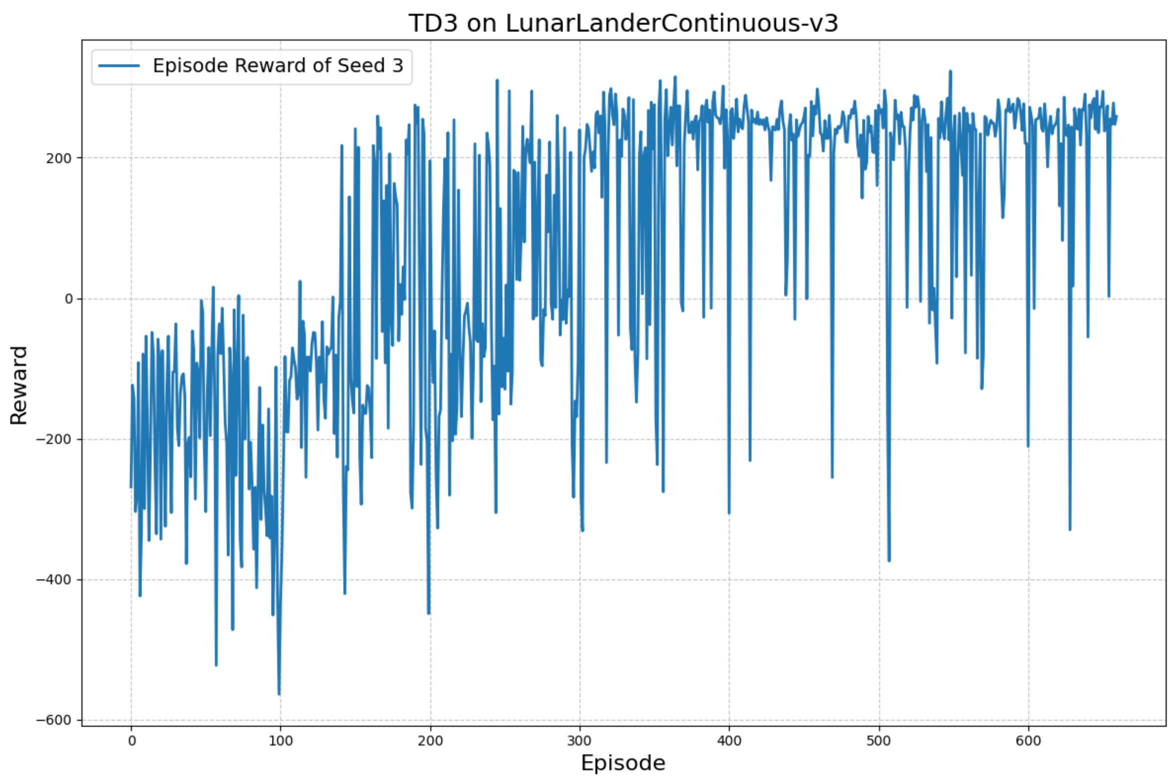

Figure 4 Episode Reward of Seed 3

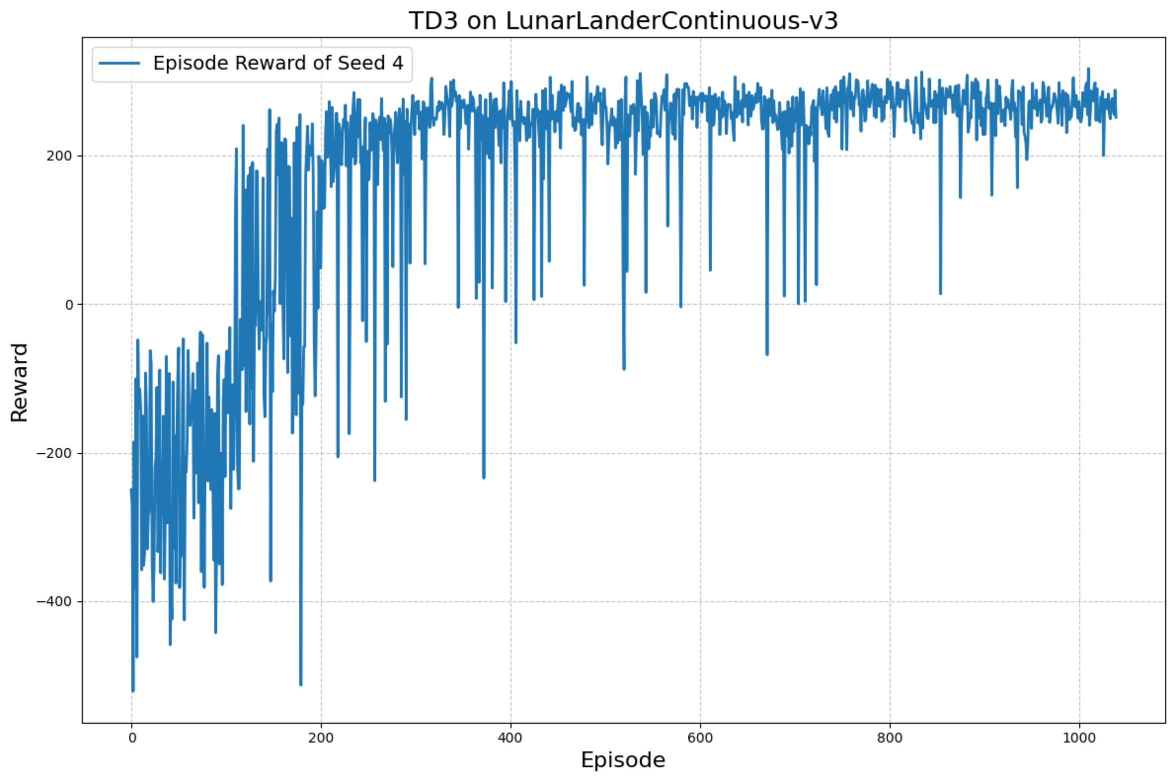

Figure 5 Episode Reward of Seed 4

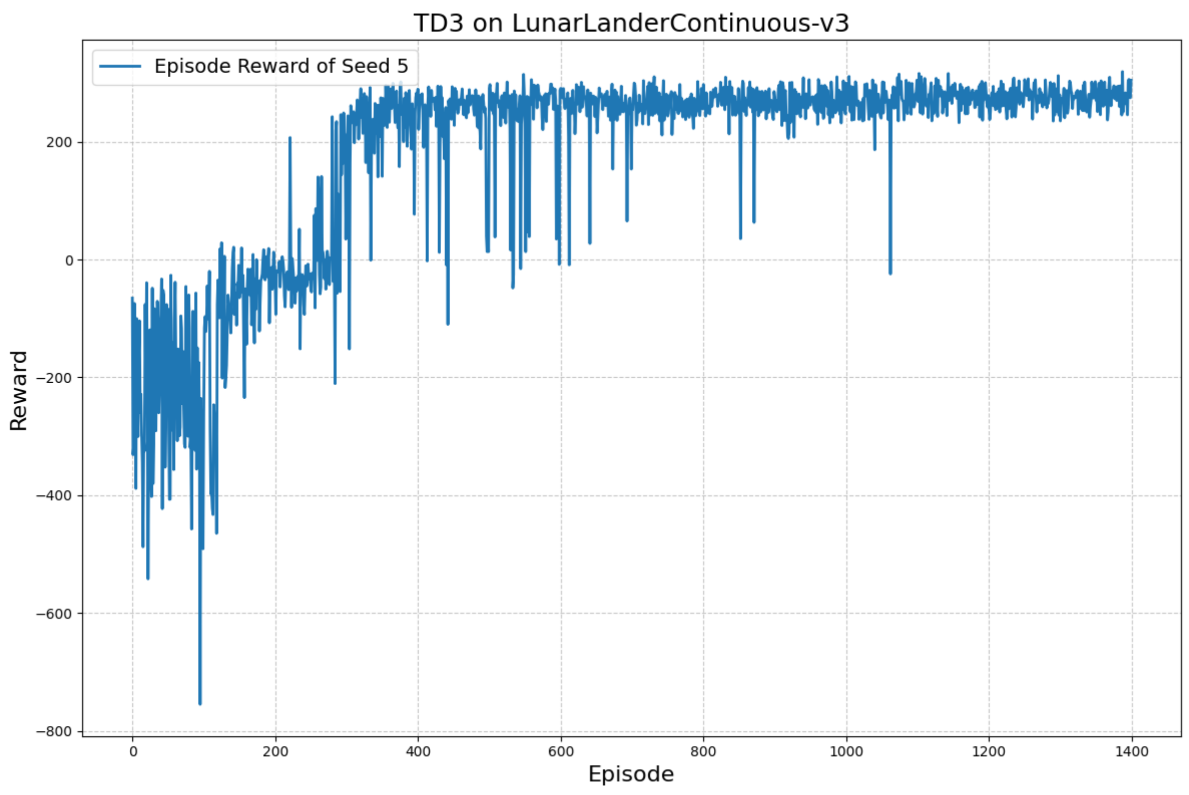

Figure 6 Episode Reward of Seed 5

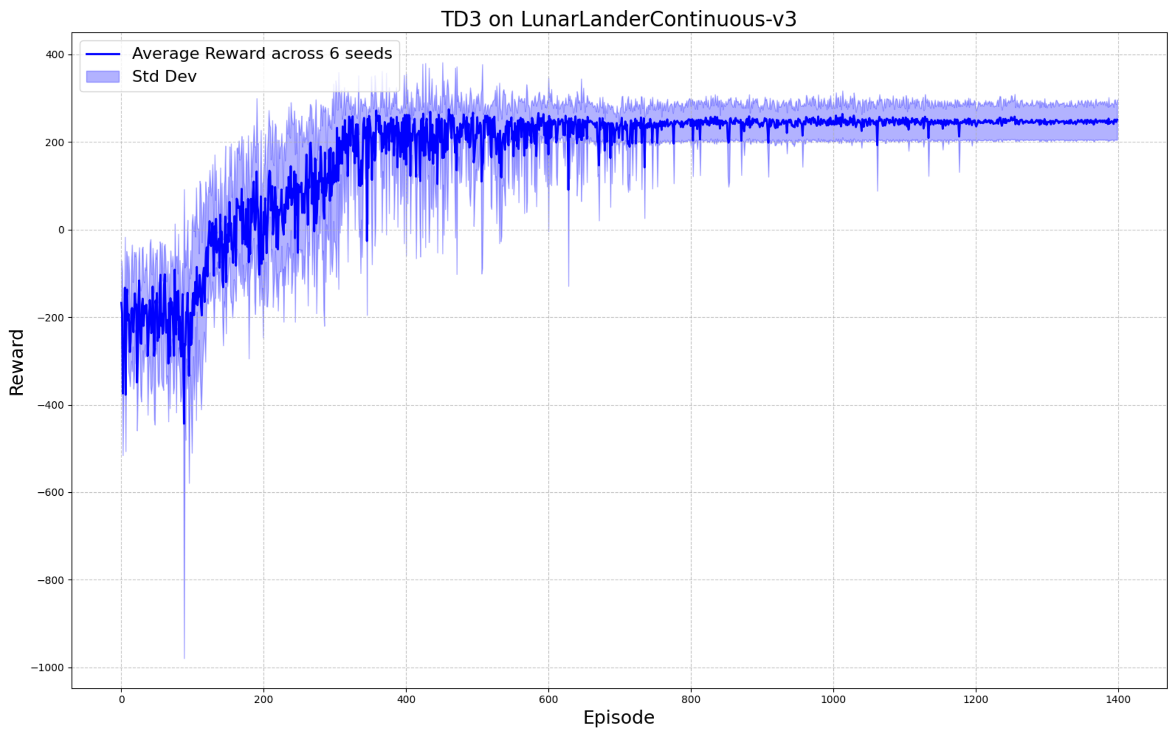

Figure 7 Average Reward Across 6 Seeds

Seed-by-Seed Analysis

Seed 0

The episodic rewards began at approximately -1000 to -1500, reflecting the randomness and lack of initial policy optimization. The rewards stabilized around 250, with slight fluctuations near convergence. A sharp upward trend was visible between episodes 0 to 200, demonstrating rapid improvement. The rewards plateaued after approximately 600 episodes, consistently exceeding the success threshold 200.

Seed 1

Initial rewards fluctuated around -600 to -200, indicating poor initial policy performance. The rewards consistently hovered around 200-250 in the later stages of training. The learning curve consistently rose over 300 episodes, with significant variability in intermediate rewards before achieving stability.

Seed 2

Initial performance varied between -600 and -400. The agent achieved rewards of approximately 250 by the end of training. Compared to other seeds, a smoother learning trajectory was observed, with the rewards stabilizing after 800 episodes.

Seed 3

Rewards started between -400 and -200, indicating better initial performance than other seeds. Converged to consistent rewards around 250, with minor fluctuations. Despite early fluctuations, the learning stabilized effectively after 400 episodes.

Seed 4

Initial rewards fluctuated widely between -600 and -200. Consistent rewards above 200 were observed in the final episodes. The rewards stabilized after 600 episodes, with the agent achieving high performance consistently.

Seed 5

The reward starts around -400. Stabilization occurs around episode 500. Rewards maintain an average of approximately 200, with periodic dips—slightly higher volatility in the mid-phase, which later stabilizes.

Table Explanation

The table above presents the best average reward achieved per 100 episodes for 6 different random seeds during the training of the TD3 agent on the LunarLanderContinuous-v3 environment. The values provide a numerical representation of the learning progression, reflecting the differences in convergence speed, stability, and final performance across seeds.

The starting rewards, observed at the 100-episode mark, vary significantly across seeds. For example, Seed 5 has the lowest average reward of -232.49, while Seed 1 starts relatively higher at -197.51. These values reflect the environment's stochastic nature and random initialization's impact on the agent's exploration and early learning dynamics.

By the 500-episode mark, all seeds achieve an average reward exceeding the threshold of 200, demonstrating convergence to a high-performing policy. Seeds like Seed 1 and Seed 5 exhibit faster improvement, achieving rewards of 249.26 and 245.00, respectively, by episode 500. Meanwhile, the table shows that some seeds, such as Seed 0, achieve similar performance levels slightly later in training. After the 1000-episode mark, most seeds stabilize with rewards exceeding 270, with Seed 2 achieving the highest value of 293.60 by episode 1100. This highlights the algorithm's ability to reach optimal performance regardless of seed-specific variability consistently.

The later rows in the table, particularly beyond 1000 episodes, indicate a plateau in performance for most seeds. Seeds like Seed 5 reach a final average reward of 285.23, while Seed 2 achieves 283.37, reflecting the robustness of the TD3 implementation in solving the environment. The variation in final performance across seeds is relatively small, suggesting that the algorithm consistently produces high-quality solutions despite differences in convergence dynamics. These results reinforce the importance of averaging across multiple seeds to provide a robust performance assessment.

Average Reward Across All Seeds

The rewards averaged around -600 at the beginning of training. The rewards stabilized around 250, with the standard deviation bands narrowing significantly in later episodes, indicating reduced variability. Initially, the variability was high, spanning various rewards across seeds. However, as training progressed, the deviation decreased significantly, reflecting the convergence of all seeds to high rewards. The agent's learning curve averaged across seeds was smooth, with minor fluctuations. By approximately 800 episodes, the rewards consistently exceeded the 200 thresholds for success.

All seeds exhibit a steep upward trajectory in the early episodes, representing rapid learning during the exploratory phase. Significant noise and variability were evident in the intermediate stages, indicative of policy refinement and environmental stochasticity. Convergence was observed across all seeds, with rewards stabilizing near the 250 marks by the end of training.

The results demonstrate that the TD3 implementation is robust and performs well across multiple random seeds. The agent consistently exceeded the success threshold of 200 rewards, with final rewards averaging approximately 250 across seeds. The standard deviation analysis further corroborates the reliability of the learned policies, as variability diminished significantly after convergence. These outcomes highlight the effectiveness of the TD3 algorithm in solving continuous action-space problems like the Lunar Lander Continuous environment.

Explanation of Differences

The differences in performance across random seeds in the TD3 experiments can be attributed to the environment's inherent stochasticity and the agent's initialization. The Lunar Lander environment introduces randomness through its physics simulation, such as wind, gravity, and initial conditions, which affect the trajectory of the lander and the rewards obtained for similar actions. This variability can lead to different learning dynamics, as some initial conditions may be more favorable for exploration and learning than others, giving the agent a more straightforward path toward optimizing its policy.

Another contributing factor to the variations is the random initialization of neural network weights in the actor and critic networks. These initializations influence the agent's early state-action values and exploration strategy predictions. If the initial weights bias the agent toward less optimal regions of the state space, it may take longer to discover high-value trajectories, resulting in slower convergence. On the other hand, a fortuitous initialization could guide the agent toward favorable state-action pairs early on, accelerating learning. This sensitivity to initialization is typical in reinforcement learning algorithms, especially in continuous action spaces.

The differences also stem from how noise is applied during the exploration phase of training. TD3 uses Gaussian noise to encourage exploration, but the effects of this noise can vary depending on the seed. Some seeds might result in the agent encountering rewarding states more frequently, reinforcing positive behavior early in training. Conversely, other seeds might lead to less effective exploration, requiring the agent to train longer to achieve comparable performance. These factors combine to create the observed variability across seeds, highlighting the importance of averaging results over multiple trials to provide a robust measure of algorithm performance.

Appendix

Model weights and episode reward results as CSV and .npy are below:

https://drive.google.com/drive/folders/1EMiiubHtTw2PP_UraZ3SK1FzDd6as4Qb?usp=sharing