2024

Ai Project

MUJOCO ANT-V4

Faculty of Engineering, Helwan University

Supervised by: Dr Ahmed Badawy

TA: Eng AbdelRahman Arabawy

Table of Contents

Team Members

1. Mostafa Adel Ali
2. Michael Raouf Eldeeb
3. Menna Mabrouk Ali
4. Marawan Ragab Mahmoud
5. Heba Elsayed Mohamed

Abstract

Reinforcement learning (RL) has emerged as a powerful paradigm for training agents to interact with complex environments and learn optimal behaviors through trial and error. In this project, we delve into the realm of RL by tackling the challenging **Mujoco Ant-v4** environment, a benchmark problem in continuous control tasks. Our objective is to train an RL agent capable of mastering the nuanced dynamics of the Mujoco Ant-v4 environment, ultimately achieving proficient locomotion and control of the ant-like robot.

To tackle this challenging task, we employ the **Twin Delayed DDPG (TD3)** algorithm, a state-of-the-art RL algorithm known for its stability and effectiveness in continuous control problems. The TD3 algorithm, which extends the popular Deep Deterministic Policy Gradient (DDPG) algorithm, offers improvements in training stability and sample efficiency, making it well-suited for our task. **Stable-baseline3** package was used to deploy the TD3 agent in our code. (OpenAi, n.d.)

Furthermore, we adopt a **Multi-Layer Perceptron (MLP)** policy architecture to represent the agent's policy function. The MLP policy provides the flexibility to learn complex mappings from observations to actions, enabling the agent to adapt to the intricate dynamics of the Mujoco Ant-v4 environment. (Stable-baseline3, n.d.)

One of the key challenges in RL training is exploration, especially in high-dimensional action spaces. To address this challenge and facilitate effective exploration during training, we incorporate external action noise using the **NormalActionNoise** and **Ornstein-Uhlenbeck Noise** mechanism. By adding Gaussian noise to the agent's actions, we encourage exploration of the action space, enabling the agent to discover optimal policies more efficiently. (OpenAi, n.d.)

Through this project, we aim to explore the capabilities of RL in solving complex continuous control tasks and contribute to the advancement of robotic locomotion and control. Our goal is to train an RL agent that demonstrates proficient navigation and control in the Mujoco Ant-v4 environment, showcasing the potential of RL in real-world applications.

Ant-v4

Env Description

The Mujoco Ant-v4 environment, originally introduced by Schulman et al. in their seminal work on high-dimensional continuous control, presents a formidable challenge to RL agents. The environment simulates a 3D robotic system comprising a torso and four articulated legs, each consisting of two body parts connected by hinges. The task for the agent is to coordinate the movements of these legs to propel the ant forward while maintaining stability and preventing it from flipping over.

Action Space

The **Action Space** of the Ant environment in the Gymnasium documentation is defined as follows:

- It's a **Box(-1.0, 1.0, (8,), float32)**, which means it's an 8-dimensional box with each dimension allowing a range of values from -1.0 to 1.0.
- Each dimension corresponds to a **torque** applied at one of the **eight hinge joints** of the Ant robot.
- The torques are applied to control the movement of the Ant's legs, with each leg having two body parts connected by these hinges.

A breakdown of the **Action Space**:

- **Num**: Index of the action.
- **Action**: The specific torque applied.
- **Control Min**: The minimum value of the torque, which is -1.
- **Control Max**: The maximum value of the torque, which is 1.
- **Name**: The name of the joint or hinge in the Ant's body to which the torque is applied.
- **Joint Unit**: The unit of the torque, which is Newton-meter (N m).

For example, the first action (Num 0) applies a torque on the rotor between the torso and the back right hip, with a minimum and maximum possible value of -1 and 1 respectively, and is named "hip_4 (right_back_leg)" in the XML file. The same structure applies to all eight actions in the space.

Observation Space

The **Observation Space** in the Gymnasium documentation for the **Ant environment** is detailed as follows:

- **Type**: It is a Box type with continuous values.
- **Dimensions**: The space has 27 dimensions if the position of the ant's torso is excluded, otherwise, it has 29 dimensions.
- **Range**: Each dimension has a range from negative infinity to positive infinity (-inf to inf).

The **Observation Space** includes:

- **Positional Values**: These are the z-coordinate and orientations (x, y, z, w) of the ant's torso, and the angles between the torso and the legs, as well as between the individual leg links.
- **Velocities**: The velocities of the torso and the angular velocities of the joints between the torso and legs and between the leg links.

A **breakdown** of the key components:

- **Torso Position and Orientation**: Includes the z-coordinate and quaternion representation of the orientation.
- **Leg Angles**: Hinge angles between the torso and leg links and between the leg links themselves.
- **Velocities**: Linear and angular velocities of the torso and angular velocities of the leg hinges.

Observation Space is crucial for the agent to perceive its environment and make decisions accordingly. It provides a comprehensive set of data points that represent the state of the ant robot in the simulation.

Rewards

- **Healthy Reward**: A fixed reward given at each timestep the ant is considered "healthy".
- **Forward Reward**: Calculated based on the ant's forward movement, specifically as the change in the x-coordinate divided by the time step dt.
- **Control Cost**: A penalty for using large actions, computed as the sum of the squares of the actions multiplied by a control cost weight.
- **Contact Cost** (if applicable): A penalty for large external contact forces, calculated similarly to control cost but with contact forces and a separate weight.

Starting State

- The ant starts with all positional values set to zero, except the z-coordinate of the torso, which is slightly elevated.
- A uniform noise is added to the positional values, and standard normal noise is added to the velocity values for randomness.
- The initial orientation is set to face forward.

Episode End

- The episode ends if the ant becomes "unhealthy" (non-finite state values or the torso's z-coordinate outside a specified range) or after 1000 timesteps.
- If terminate_when_unhealthy is True, the episode can end prematurely if the ant is unhealthy.
- If terminate_when_unhealthy is False, the episode only ends after reaching the maximum timestep limit.

These components are crucial for training reinforcement learning agents as they define the objectives, initial conditions, and end conditions for the simulation. (Gymnasium, Farma, n.d.)

TD3 Agent

Background

While DDPG can achieve great performance sometimes, it is frequently brittle with respect to hyperparameters and other kinds of tuning. A common failure mode for DDPG is that the learned Q-function begins to dramatically overestimate Q-values, which then leads to the policy breaking, because it exploits the errors in the Q-function. Twin Delayed DDPG (TD3) is an algorithm that addresses this issue by introducing three critical tricks:

Trick One: Clipped Double-Q Learning. TD3 learns *two* Q-functions instead of one (hence “twin”) and uses the smaller of the two Q-values to form the targets in the Bellman error loss functions.

Trick Two: “Delayed” Policy Updates. TD3 updates the policy (and target networks) less frequently than the Q-function. The paper recommends one policy update for every two Q-function updates.

Trick Three: Target Policy Smoothing. TD3 adds noise to the target action, to make it harder for the policy to exploit Q-function errors by smoothing out Q along changes in action.

Together, these three tricks result in substantially improved performance over baseline DDPG. (OpenAi) (Antonin Raffin, n.d.)

Exploration vs Exploitation

TD3 trains a deterministic policy in an off-policy way. Because the policy is deterministic, if the agent were to explore on-policy, in the beginning it would probably not try a wide enough variety of actions to find useful learning signals. To make TD3 policies explore better, we add noise to their actions at training time, typically uncorrelated mean-zero Gaussian noise.

At test time, to see how well the policy exploits what it has learned, we do not add noise to the actions. (OpenAi, n.d.)

External Action Noises

Because the policy is deterministic, DDPGand TD3 rely on external noise for exploration. Exploration is crucial for discovering effective strategies. **Deterministic policies**, like those used in DDPG (Deep Deterministic Policy Gradient) and TD3 (Twin Delayed DDPG), select actions without any randomness, which can limit exploration. To counter this, **external noise** is added to the actions during training. This noise encourages the policy to explore a wider range of actions, leading to a more robust learning process. In TD3, specifically, exploration is enhanced by adding Gaussian noise to the policy's actions, which helps in preventing premature convergence to suboptimal policies and promotes the discovery of more diverse and potentially better solutions.

We tested the agent on two external noises:

1. Normal Action Noise
2. Ornstein-Uhlenbeck Noise

After training our agents with both action noises, experiments demonstrate that the model trained with **Ornstein-Uhlenbeck** action noise outperforms the model trained with **NormalActionNoise**. Across ten episodes of evaluation, the Ornstein-Uhlenbeck model achieves a higher mean score of approximately **3076.07** compared to 2888.23 for the NormalActionNoise model. Notably, the Ornstein-Uhlenbeck model reaches its performance level in approximately **700,000** timesteps, whereas the NormalActionNoise model requires around **1 million** timesteps to achieve its level of performance. These results highlight the effectiveness of Ornstein-Uhlenbeck action noise in facilitating faster learning and superior performance in the Mujoco Ant-v4 environment.

Normal Action Noise

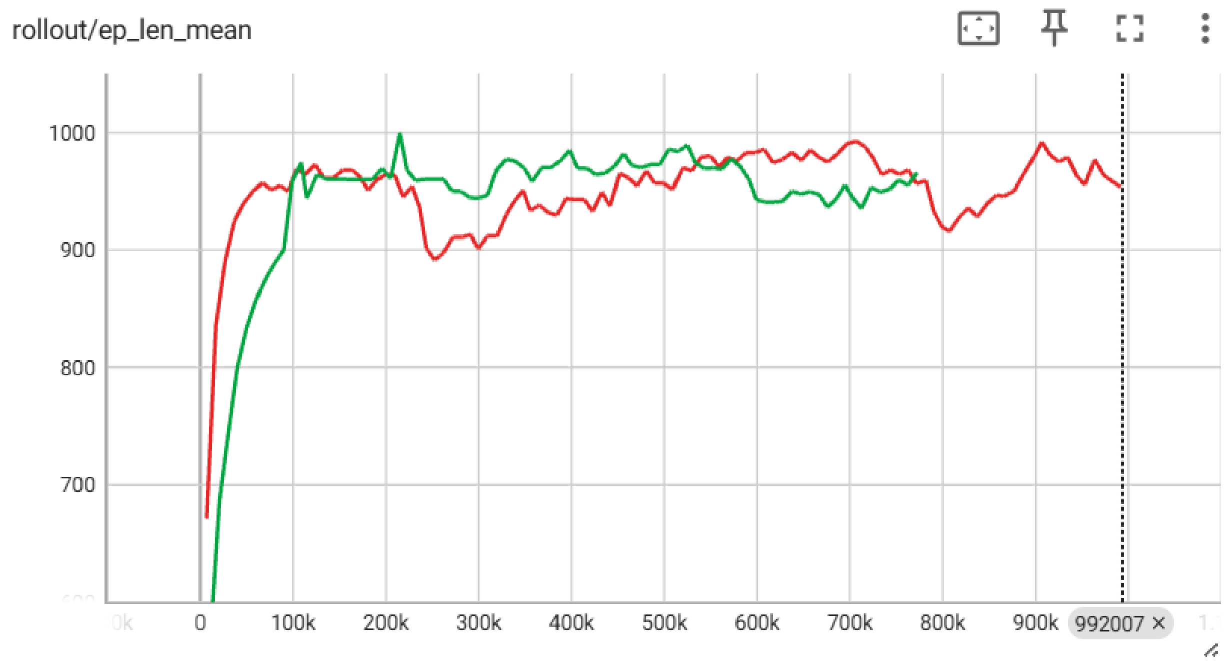
1. **What is NormalActionNoise?**
    - **NormalActionNoise** is a type of exploration noise commonly used in reinforcement learning algorithms, such as TD3 (Twin Delayed DDPG).
    - It adds independent Gaussian noise to each action component.
2. **How Does it Work?**
    - At each time step, Gaussian noise is sampled independently for each action dimension from a normal distribution with a mean of zero and a user-defined standard deviation.
    - This noise introduces randomness to the agent's actions, encouraging exploration of the action space.
3. **Why Choose NormalActionNoise for Exploration?**
    - NormalActionNoise is straightforward and computationally efficient.
    - It's suitable for environments where actions are relatively independent of each other and there's no inherent temporal correlation in exploration.
    - In simpler environments or those with less complex dynamics, where actions do not require smoothness or temporal correlation, NormalActionNoise can be an effective choice.
4. **Why NormalActionNoise for the Ant-v4 Environment?**
    - While Ornstein-Uhlenbeck noise is recommended for environments with complex dynamics and where smooth, correlated exploration is beneficial, NormalActionNoise can also be considered.
    - In environments like Ant-v4, where the dynamics are relatively simpler compared to some other MuJoCo environments, NormalActionNoise may suffice.
    - Additionally, NormalActionNoise might be preferred if computational efficiency is a concern or if the exploration requirement is not highly correlated or smooth actions.

Ornstein-Uhlenbeck Noise

1. **What is Ornstein-Uhlenbeck Noise?**
    - Ornstein-Uhlenbeck noise is a stochastic process named after physicists Leonard Ornstein and George Uhlenbeck. It's commonly used in physics and engineering to model random fluctuations that have some persistence or correlation over time.
    - In simpler terms, Ornstein-Uhlenbeck noise adds randomness to actions in a way that makes nearby actions more likely to be similar. It introduces a sense of smoothness or continuity in the exploration process.
2. **How Does it Work?**
    - Ornstein-Uhlenbeck noise is characterized by a mean value, a strength parameter, and a volatility parameter.
    - At each time step, the noise is drawn from a normal distribution with a mean of zero. Then, a portion of the previous noise value is added to the current value, creating temporal correlation.
    - Essentially, it adds a memory effect to the noise, where the noise at one time step depends partially on the noise at the previous time step.
3. **Why Ornstein-Uhlenbeck Noise for the Ant-v4 Environment?**
    - In complex environments like the MuJoCo Ant-v4, where the agent controls a dynamic and multi-joint robotic system, actions often need to be smooth and continuous for effective exploration.
    - Ornstein-Uhlenbeck noise provides this smoothness by encouraging nearby actions to be similar, helping the agent explore the action space more efficiently.
    - This type of noise is particularly useful when actions should exhibit inertia or when exploration needs to have memory of past actions, which is often the case in robotic control tasks.

In summary, Ornstein-Uhlenbeck noise adds a smooth and temporally correlated randomness to the agent's actions, making it a suitable choice for exploration in complex and dynamic environments like the MuJoCo Ant-v4, where smooth and continuous actions are essential for effective learning and control.

Learning Graphs

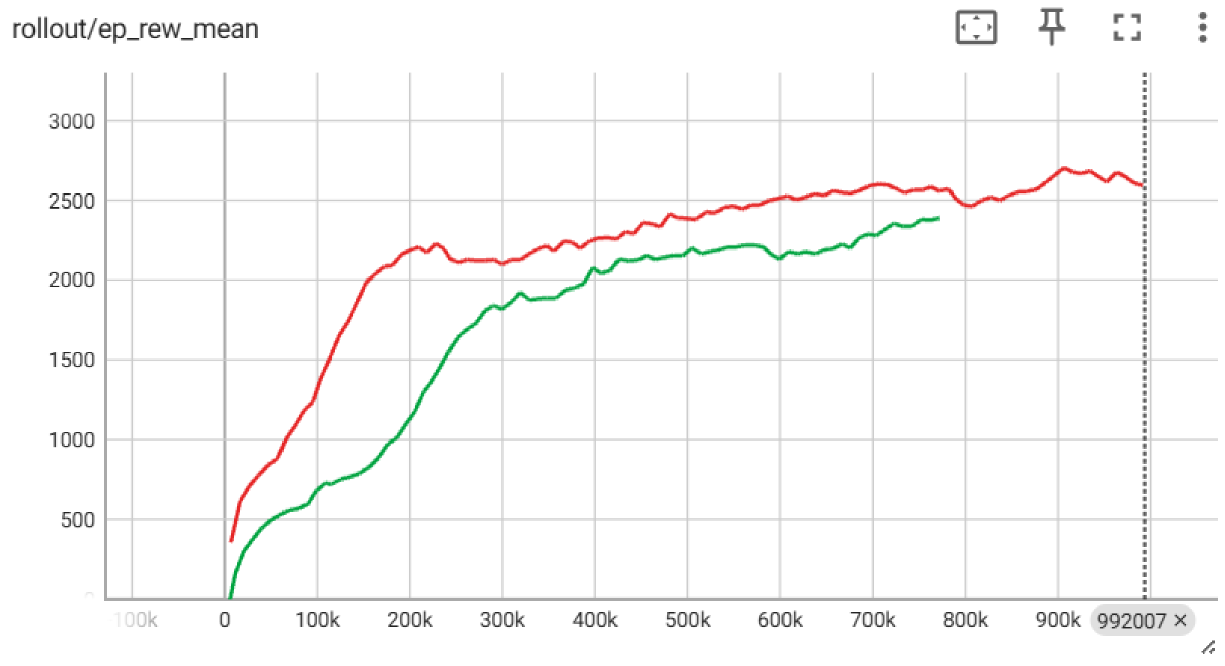


	Run ↑	Value	Step	Relative
🔴	TD3_NA	953.52	992,007	3.121 hr
🟢	TD3_Orn	965.8	772,544	2.282 hr

rollout means are plotted every **100** episodes.

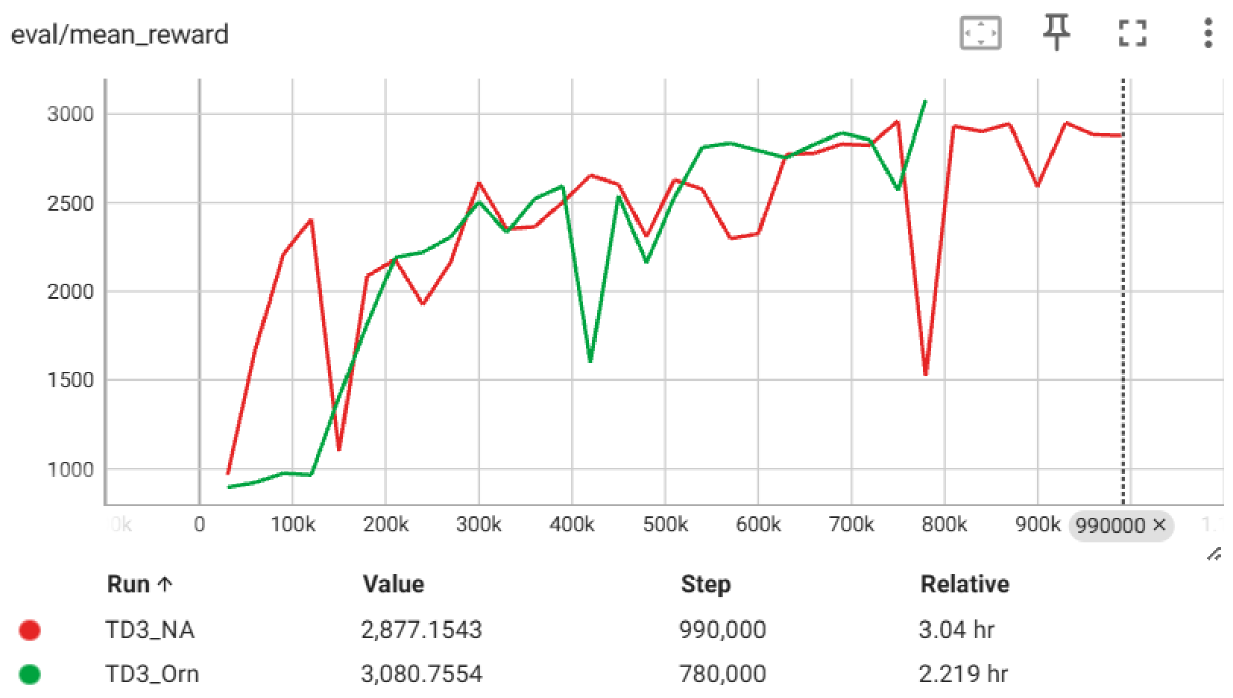
Episode length mean caps at **1000** (*truncated state*). *Terminated state* is when the Ant agent is *unhealthy* (flipped). So, the episode ends prematurely.

The graphs show that both agents learn to prolong the episode quite early. But when compared to the **rollout reward means**, it's apparent that the model has yet to learn achieving satisfactory velocities in the early stages.



	Run ↑	Value	Step	Relative
🟥	TD3_NA	2,593.0974	992,007	3.121 hr
🟩	TD3_Orn	2,390.0288	772,544	2.282 hr

Both **rollout reward means** converge later on at satisfactory rewards. However, it seems that the NA has better performance than Ornstein. But that is only in the learning process. However, in actual testing, Ornstein yields much better performance consistently (*refer to next graph for actual evaluation*).



	Run ↑	Value	Step	Relative
🟥	TD3_NA	2,877.1543	990,000	3.04 hr
🟩	TD3_Orn	3,080.7554	780,000	2.219 hr

The models are evaluated every 30k steps. When a reward threshold of 3000 is hit, the training process stops. The Ornstein model hit the threshold at relatively early stage, while the NA never hit the threshold.

References

Antonin Raffin, J. K. (n.d.). Retrieved from Smooth Exploration for Robotic Reinforcement Learning: https://arxiv.org/abs/2005.05719

Gymnasium, Farma. (n.d.). Retrieved from Ant-v4: https://gymnasium.farama.org/environments/mujoco/ant/

OpenAi. (n.d.). *TD3*. Retrieved from https://spinningup.openai.com/en/latest/algorithms/td3.html

Stable-baseline3. (n.d.). Retrieved from MlpPolicy: https://stable-baselines3.readthedocs.io/en/master/guide/custom_policy.html