ASSIGNMENT 3 REPORT
PART 2

Part II [50 points] - Implementing Advantage Actor Critic (A2C/A3C)

1. **Discuss:**

What are the roles of the actor and the critic networks?

Sol.

In A2C, two networks work together to learn an optimal policy:

1. Actor

It is responsible for selecting actions. It outputs the probability distribution over actions, typically through the mean and standard deviation of a Gaussian distribution. It updates its policy to maximize expected returns, guided by the critic's feedback.

2. Critic

The critic estimates the value function V(s), which predicts how good a given state is. It provides feedback on how good the taken actions were, helping the actor make better decisions.

The actor improves the policy, and the critic evaluates it.

- **What is the "advantage" function in A2C, and why is it important?**

The advantage function measures how much better an action is compared to the average $A(s,a)=Q(s,a)-V(s)$

In A2C, approximating the advantage function A(s,a) as $R_t-V(s)$ is vital. This technique effectively reduces the variance in policy gradient updates, leading to more stable learning. Furthermore, it offers the actor nuanced feedback by indicating if a reward surpasses or falls short of the expected value V(s), rather than just its absolute magnitude. This centered perspective accelerates and stabilizes the convergence of the learning process.

- **Describe the loss functions used for training the actor and the critic in A2C**

A2c uses two loss functions one for the actor and one for the critic and this comes to total loss

Actor loss:

$$L_{actor}=-\log\pi(a|s)\cdot A(s,a)$$

encourages the policy to favor actions with positive advantage. The negative sign in policy gradient updates arises from the use of a minimization framework to achieve the goal of maximizing the expected return.

Critic Loss:

$$L_{critic}=MSE(V(s),R_t)$$

Training the critic involves minimizing the discrepancy between its predicted value V(s) and the actual return Rt. This minimization process enables the critic to make more accurate value estimations, which in turn leads to a more reliable calculation of the advantage function $A(s,a) \approx Rt - V(s)$.

Entropy Bonus:

Q $L_{entropy} = -\beta \cdot H[\pi(\cdot|s)]$

Maximizing the policy's entropy encourages the agent to explore a wider range of actions. This exploration helps prevent the policy from prematurely converging to a suboptimal deterministic strategy.

Total Loss $L = L_{actor} + 0.5 \cdot L_{critic} - \beta \cdot L_{entropy}$

2. **Briefly describe the environment that you used (e.g. possible actions, states, agent, goal, rewards, etc). You can reuse related parts from your previous assignments.**

We used Cartpole-v1 environment from Gymnasium, it is a continuous control task where we train a pole to stand upright for certain timestamps.

Possible actions:
The agent can choose between two discrete actions—move the cart left (0) or right (1)

States:
The state space is a 4-dimensional vector representing:

Cart position – min: -4.8 max: 4.8

Cart velocity, – min: -inf max: inf

Pole angle, – min: -24 degree max: 24 degree

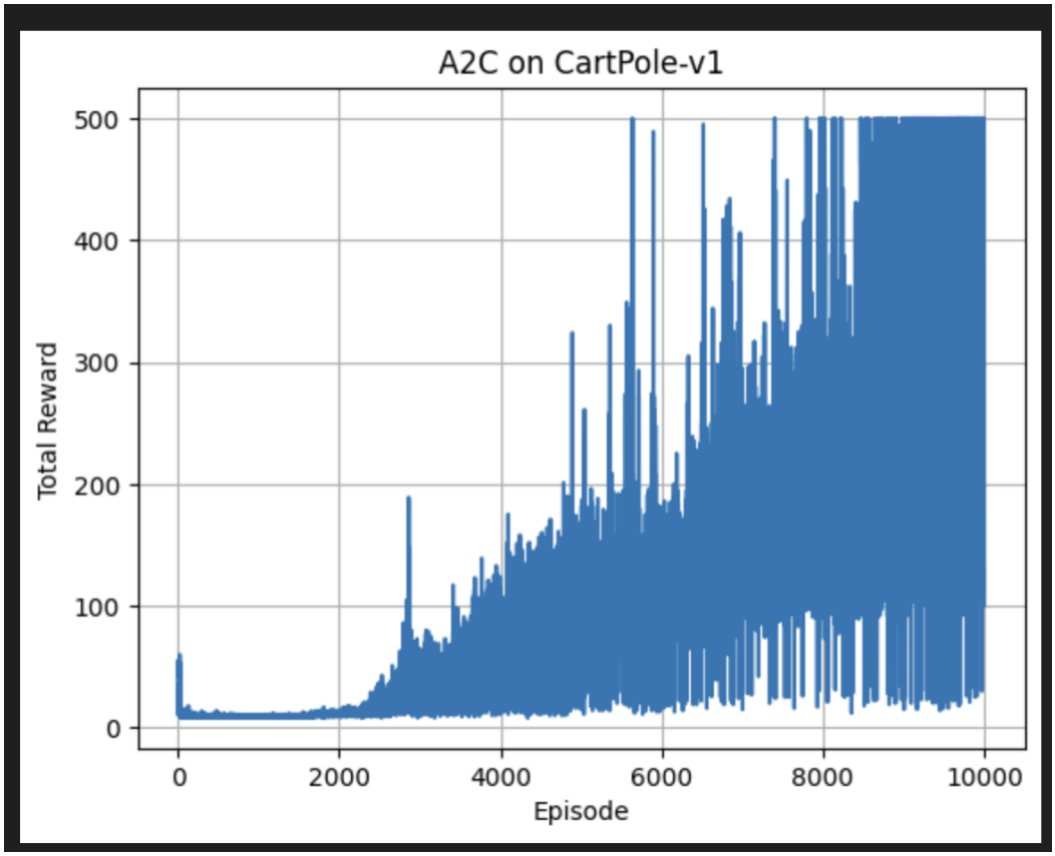
Pole angular velocity. – min: -inf max: inf

Agent:
The agent learns a policy to balance the pole by choosing actions based on the current state.

Goal**:**
Keep the pole balanced upright (i.e., prevent it from falling past a certain angle) for as long as possible.

Rewards:
The agent receives a reward of +1 for every time step the pole remains balanced. An episode ends if the pole falls beyond a threshold angle or the cart moves out of bounds. The maximum episode length is 500 time steps.

3. **Show and discuss your results after applying your A2C/A3C implementation on the environments. Plots should include epsilon decay and the total reward per episode.**



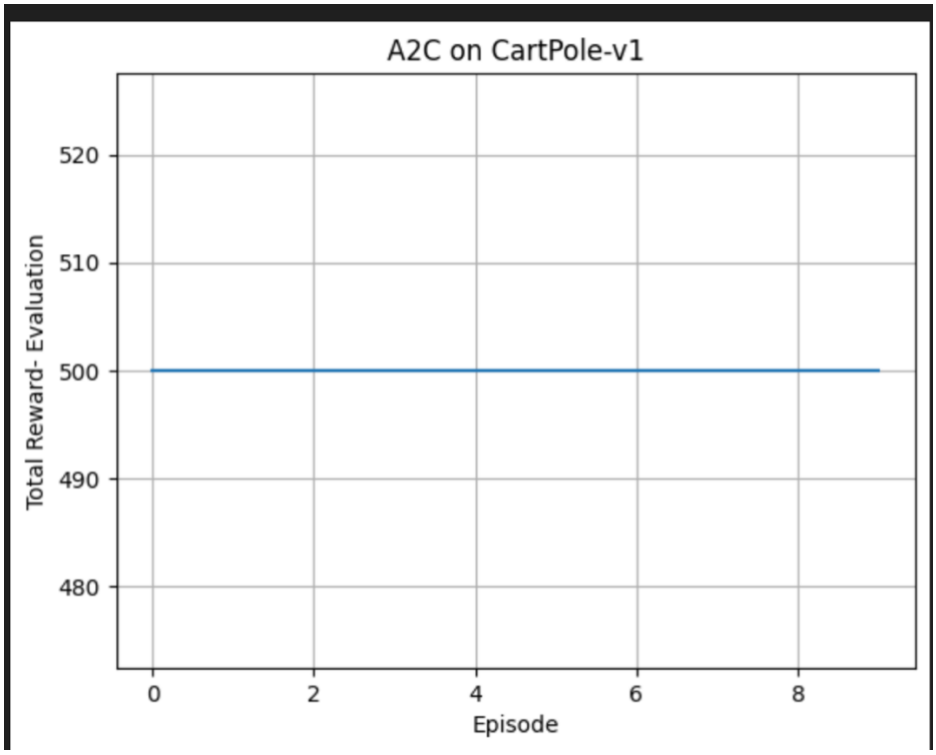
The training plot shows the performance of the A2C (Advantage Actor-Critic) agent on the CartPole-v1 environment over 10,000 episodes. The agent initially struggles, frequently falling quickly and receiving low rewards, often between 0 and 20. This behaviour reflects the agent's early random exploration, where actions are uncoordinated and ineffective.

Around episode 3,000, the agent begins to exhibit a learning curve, gradually improving its ability to balance the pole. The rewards increase steadily as it learns to keep the pole upright for longer durations.

By episode 10,000, the agent consistently achieves high rewards, often nearing the maximum of 500, indicating that it has effectively learned a stable control policy.

Entropy Regularization
A2C promotes exploration using entropy regularization, rather than an epsilon-greedy strategy. In this experiment, an entropy coefficient of $\beta = 0.005$ was used. This term encourages the policy to maintain randomness in its action selection early in training, preventing premature convergence to suboptimal deterministic policies. As training progresses, the policy becomes more confident, reducing exploration while preserving learned behaviours.

4. **Provide the evaluation results. Run your agent on the environment for at least 10 episodes, where the agent chooses only greedy actions from the learnt policy. Plot should include the total reward per episode.**



5. Provide your interpretation of the results.

The A2C agent demonstrated optimal performance on the CartPole-v1 environment, consistently achieving the maximum possible reward of 500 in all 10 evaluation episodes. This indicates that the agent has successfully learned a stable and robust policy capable of keeping the pole balanced for the entire episode duration.

CartPole-v1 has relatively low complexity and deterministic dynamics, which the agent seems to have mastered effectively. The flat reward curve at 500 also suggests that the policy generalizes well and performs reliably across evaluations.

In contrast to previous evaluations in environments with uneven terrain or stochastic elements—where performance varied and average rewards were lower—this result confirms that the current policy is highly effective and fully converged for the CartPole-v1 task.

6. Include all the references that have been used to complete this part

1. https://pytorch.org/docs/stable/multiprocessing.html
2. Lecture Slides and A3 Description
3. https://www.gymlibrary.dev/environments/classic_control/cart_pole/
4. https://medium.com/sciforce/reinforcement-learning-and-asynchronous-actor-critic-agent-a3c-algorithm-explained-f0f3146a14ab
5. https://arxiv.org/abs/1707.06347
6. https://arxiv.org/abs/1602.01783

PART 3

1. **Briefly describe the environment that you used (e.g. possible actions, states, agent, goal, rewards, etc). You can reuse related parts from your previous assignments.**

We used BipedalWalker-v3 environment from Gymnasium, it is a continuous control task where we train a two legged robot to walk across uneven terrain.

State space:

24-dimensional continuous vector representing: • LIDAR-based terrain information,
• hull angle and angular velocity,
• joint angles, velocities,

• leg-ground contacts, and body velocity.

Action Space:
4-dimensional continuous vector in the range [-1, 1]
Controls the torques applied to the hip and knee joints of the two legs.

Agent: A bipedal robot with two legs, each having two motorized joints (hip and knee). Goal: The agent must learn to walk as far as possible to the right without falling or stalling. Rewards: Positive reward for moving forward smoothly.

Penalties for:

- Excessive torque usage (to promote energy efficiency),
- Falling (ends the episode with a large negative reward),
- Standing still (encourages continuous movement).

Episode Termination:

- The robot falls,
- It completes the terrain,
- Or exceeds the maximum number of timesteps (typically 1600 steps).

2. **Show and discuss your results after training your Actor-Critic agent on each environment. Plots should include the reward per episode for TWO environments. Compare how the same algorithm behaves on different environments while training**.

Bi-pedal Walker

At the start, the agent struggles to walk, often falling quickly and receiving negative rewards between -100 and -200. This is due to random, uncoordinated movements as it explores the environment.

Around episode 3,000, the agent begins to learn basic walking behavior, resulting in a gradual increase in rewards.

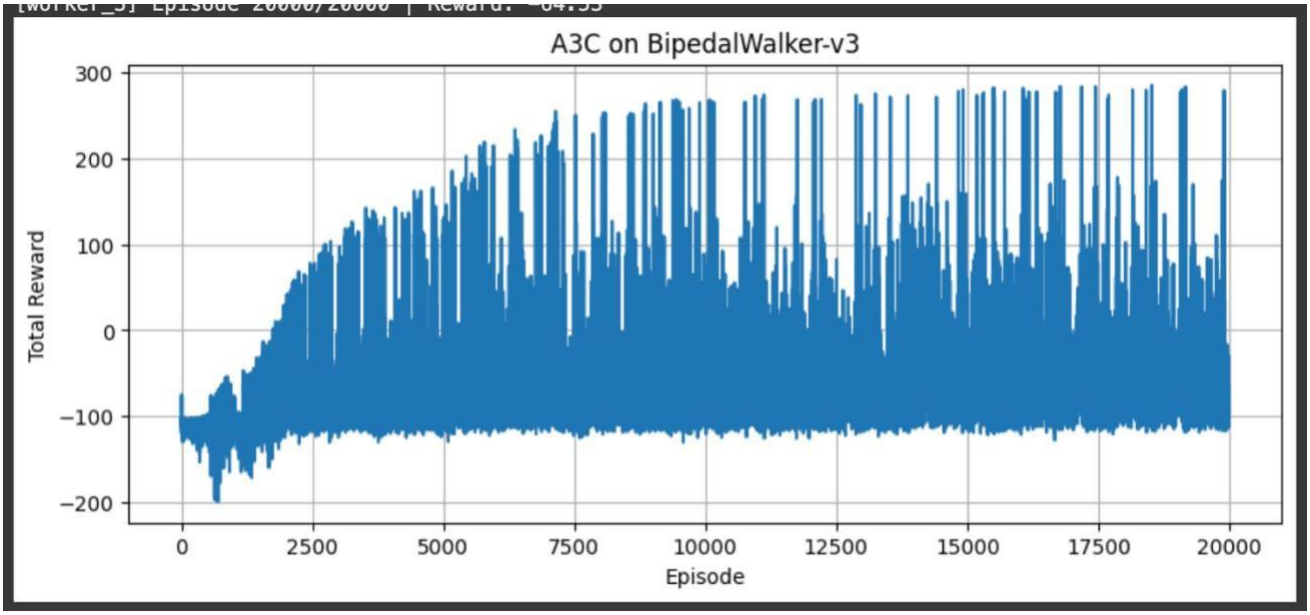
By 10,000 episodes, the agent consistently achieves high rewards (>250), indicating effective learning and stable walking.

However, the reward curve remains somewhat volatile, which is expected due to the stochastic nature

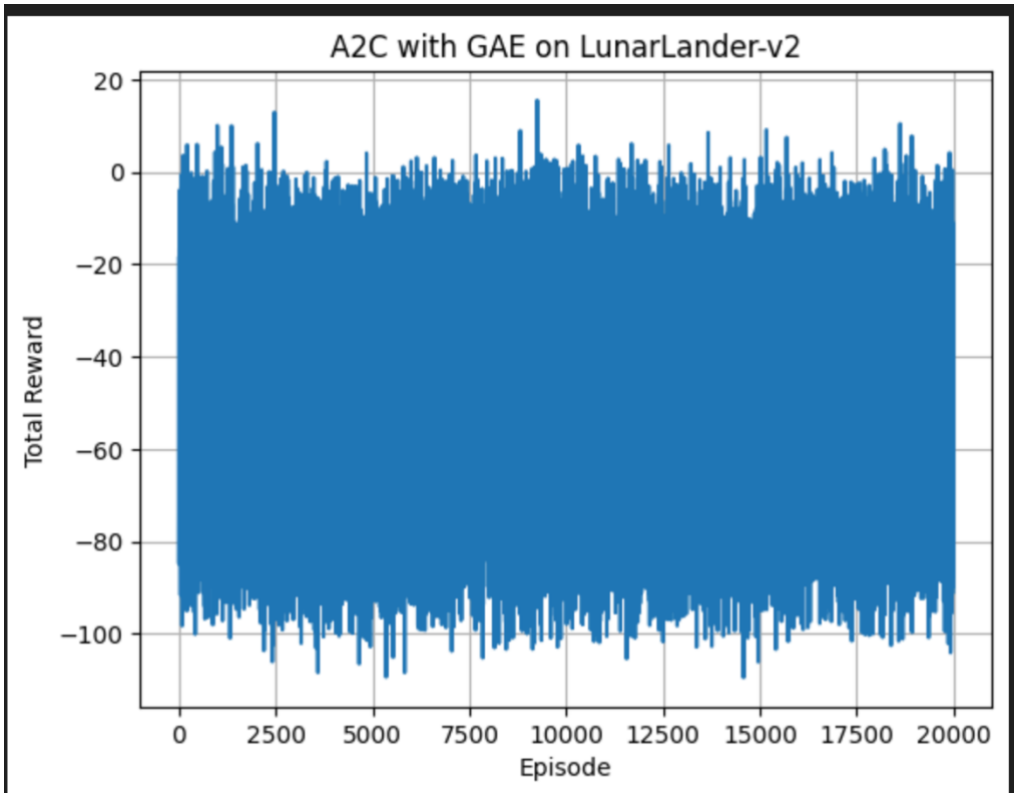
of action sampling in the policy. Occasional performance drops are part of the exploration-exploitation dynamics.

Entropy Regularization in A2C

A2C does not utilize an explicit **epsilon-greedy** strategy, it encourages exploration through entropy regularization. This mechanism prevents the policy from becoming overly deterministic too early in training. We used an entropy coefficient of ß = 0.005.



Lunar Lander:



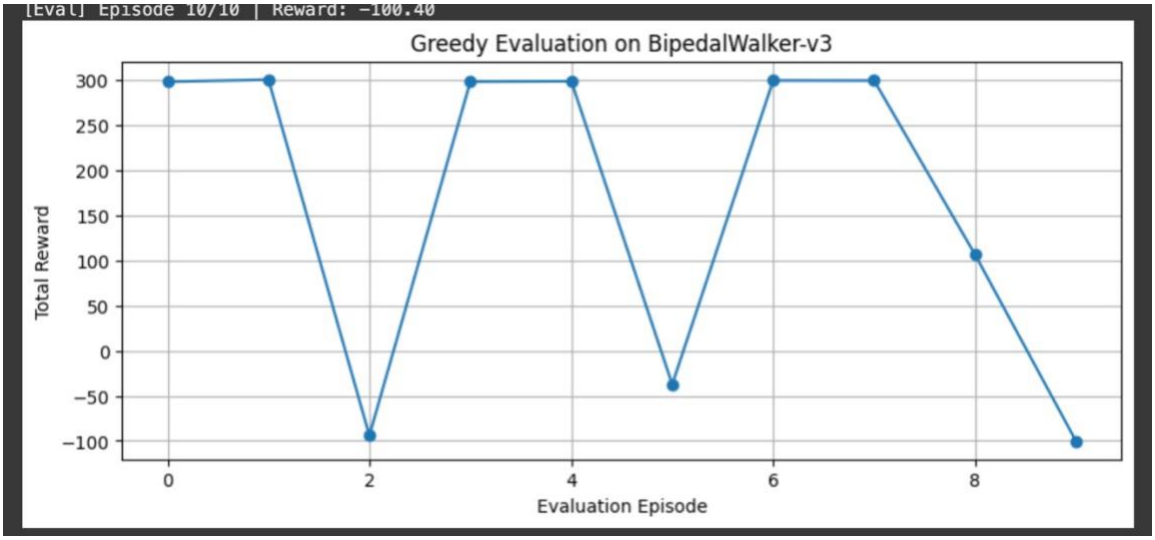
At the start, the agent struggles to land successfully, often crashing and receiving negative rewards around -100, due to unstable or random actions during initial exploration.

Throughout training, particularly across all 20,000 episodes, the agent does not exhibit consistent improvement — the rewards remain volatile and mostly negative, typically fluctuating between -100 and 0, suggesting unstable or ineffective learning.
This could indicate issues such as suboptimal hyperparameters, inadequate network architecture, or insufficient reward shaping. The absence of reward growth trends shows that the agent has not yet developed a reliable landing policy.

Exploration is maintained due to entropy regularization (with β = 0.005), which encourages the policy to remain stochastic and avoid premature convergence to suboptimal behaviour.

3. **Provide the evaluation results for each environments that you used. Run your environments for at least 10 episodes, where the agent chooses only greedy actions from the learnt policy. Plot should include the total reward per episode.**

Bi pedal Walker

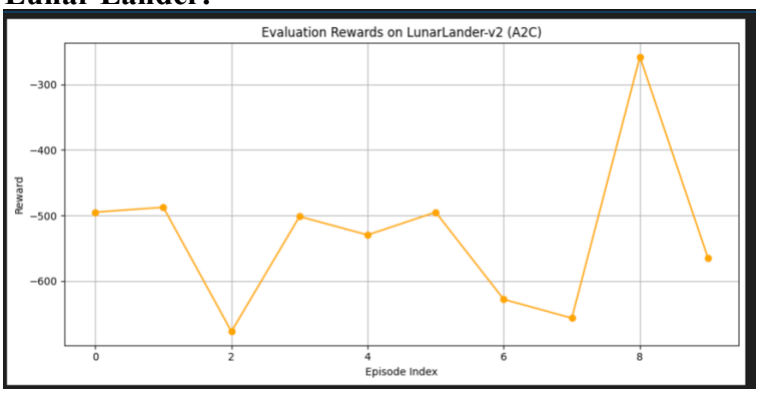


The agent demonstrated strong performance, with 6 out of 10 episodes scoring above 290, nearing the environment's maximum reward of ~300. This indicates that the learned policy is highly effective in handling typical terrain conditions.

However, episodes 3, 6, and 10 resulted in notably lower scores, suggesting that the policy can still fail in challenging scenarios—possibly due to difficult terrain configurations or the lack of exploration during greedy evaluation.

The average reward across all 10 evaluation runs was approximately **237.57**, confirming that the agent has learned to consistently walk and maintain balance, even on uneven surfaces.

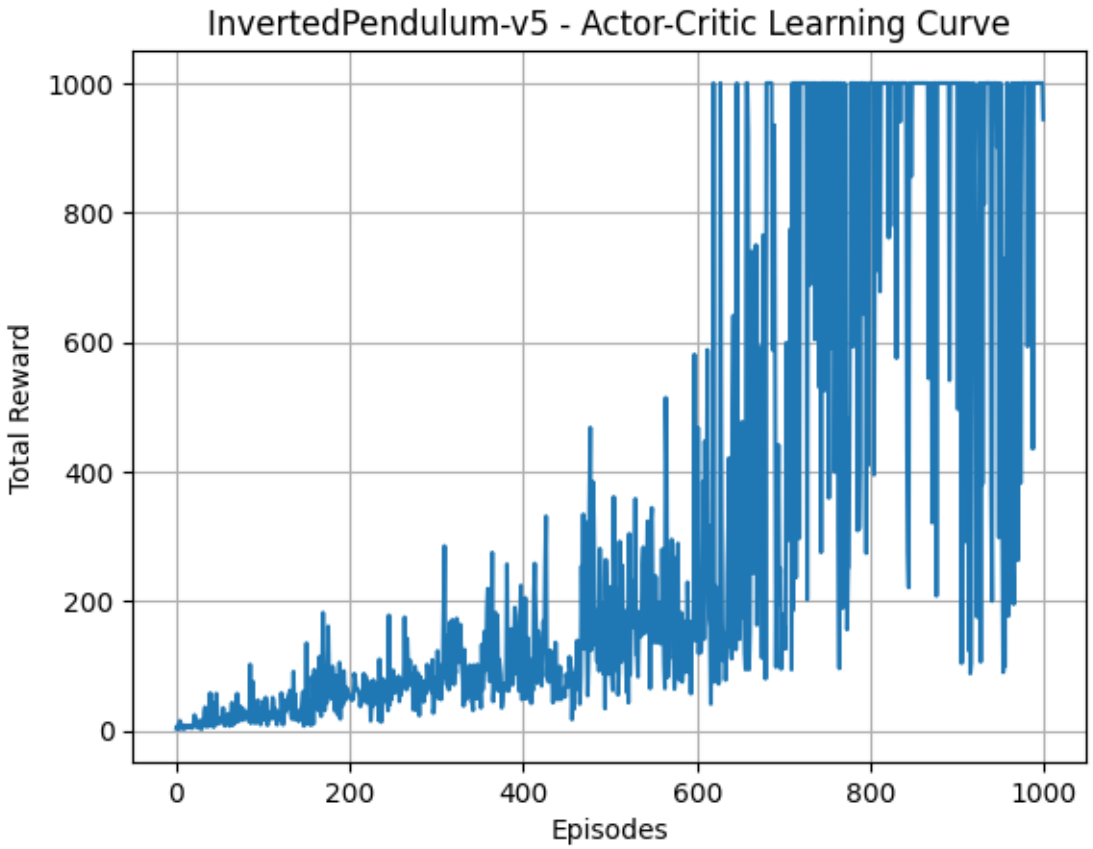
Lunar Lander:



BONUS PART

MuJoCo Environment

• Present a reward dynamic plot showing the learning curve of your agent on the MuJoCo environment.
• Describe the specific MuJoCo environment you chose and the Actor-Critic algorithm you implemented.
• Provide an analysis of the results, discussing the performance achieved and any challenges encountered during training.



Reward Dynamics Plot - The above figure shows the learning curve of the Actor-Critic agent trained on InvertedPendulum-v5 over 1000 episodes. The Y-axis represents the total reward per episode, while the X-axis represents training episodes. In the initial episodes, the rewards were low, which was indicating that the agent was unable to balance the pendulum for a long time, but thereafter, episode ~300, the reward had steadily increased. It reached the maximum around the 600 episode mark and consistently scored the maxi reward from there, showing a clear learning curve and possible convergence.'
Environment Description - The chosen MuJoCo environment for this task is InvertedPendulum-v5 which is a classic task where the agent must balance a pole on a cart by applying continuous forces. The below specifications are

- Observation Space (4-dimensional): Position, velocity, angle, and angular velocity of the pole.
- Action Space (1-dimensional): A continuous force applied to the cart.
- **Algorithm Implemented -** The algorithm used was a Vanilla Actor-Critic method with an Actor neural network that outputs a mean and standard deviation of a Gaussian distribution over continuous actions and a separate Critic neural network that estimates the value function for each state. The main training involved calculating discounted returns and advantage estimates to update the actor and the critic separately. The updation via policy gradient loss for the actor and the MSELoss via critic are updated separately.
- Few challenges we faced while completing this task was the episode termination problem. This was solved by capping off a limit on the max steps per episode where, in the case of inverted

pendulum v5 was around the 1000 limit mark. Gradient clipping was used as a safety measure for preventing gradients from becoming too large via max_norm = 0.5. Whhat we observed was that the environment in itself was highly stochastic because of the continuous action space, which was also depicted in the performance graph.

CONTRIBUTION TABLE:

Name	Part	Contribution
Pavithran Gnanasekaran	1,2,3 and bonus	50%
Rahul Ekambaram	1,2,3 and bonus	50%

REFERENCES:

1. https://pytorch.org/docs/stable/multiprocessing.html
2. Lecture Slides and A3 Description
3. https://www.gymlibrary.dev/environments/box2d/lunar_lander/
4. https://medium.com/sciforce/reinforcement-learning-and-asynchronous-actor-critic-agent-a3c-algorithm-explained-f0f3146a14ab
5. https://arxiv.org/abs/1707.06347
6. https://arxiv.org/abs/1602.01783