CSE 446 - REINFORCEMENT LEARNING - ASSIGNMENT 3

1. Discuss
    a. What are the roles of the actor and the critic networks?

Roles of the Actor and the Critic Networks
The A2C (Advantage Actor-Critic) algorithm, two distinct components, the actor and the critic, play important roles:

Actor: The actor is responsible for selecting actions based on the current policy. It maps the environment's state to a probability distribution over the possible actions, from which actions are sampled. The actor updates its policy to maximize the expected future reward by taking actions that lead to higher rewards. Essentially, the actor determines which actions to take at each step.

Critic: The critic evaluates the actions taken by the actor. It does this by estimating the value function, which represents the expected future return (reward) from a given state. The critic helps the actor by providing feedback on how good or bad the actions were, allowing the actor to refine its policy. It estimates the value of states or state-action pairs, giving the actor a measure of the desirability of its actions.

The actor and critic work together in the A2C framework: the actor explores the environment by selecting actions, while the critic evaluates those actions to guide the actor's policy improvements.

    b. What is the "advantage" function in A2C, and why is it important?

The "Advantage" Function in A2C and Its Importance
The advantage function in A2C is used to improve the training efficiency of the actor. It is defined as the difference between the Q-value (expected return) and the value function (state value), i.e., the advantage of an action in a particular state. Mathematically, it is expressed as:

$A(s,a) = Q(s,a) - V(s)$
Where:

$A(s,a)$ is the advantage function for a state s and action a

$Q(s,a)$ is the action-value function, representing the expected return from taking action

$V(s)$ is the value function, representing the expected return from state s (the critic's estimate of the state's value).

Importance of the Advantage Function:

Stabilization of Training: The advantage function plays a key role in making the policy gradient updates more stable and efficient. Without it, the updates would rely on absolute values, which can be noisy and less effective. By introducing the advantage, the actor is updated in a way that emphasizes how much better (or worse) certain actions are compared to others, making the learning process smoother and more focused. This helps the model learn more effectively and efficiently by fine-tuning the policy based on the relative value of actions rather than just their raw outcomes.

Reduced Variance: The advantage function helps smooth out the learning process by focusing on the difference between how good an action is and how good the state is, instead of just looking at the raw reward. By doing this, it reduces the impact of rewards that vary wildly, making the learning more stable and efficient. This way, the model can make more reliable updates and improve faster.

c. Describe the loss functions used for training the actor and the critic in A2C.

In A2C, the actor and critic networks are trained simultaneously with different loss functions:

Actor Loss: The actor loss is based on the policy gradient method and is defined as the negative of the log probability of the taken action multiplied by the advantage. This encourages the actor to take actions that lead to higher rewards (positive advantage) and avoid actions that lead to lower rewards (negative advantage). The formula is:

$$\mathcal{L}_{\text{actor}} = -\log(\pi(a_t|s_t)) \cdot A(s_t, a_t)$$

Where:

$\pi(a_t|s_t)$ is the probability of taking action a_t at state s_t under the current policy,

$A(s_t, a_t)$ is the advantage, which indicates how good the action a_t is compared to the baseline value $V(s_t)$

Critic Loss: The critic's loss function is typically based on the mean squared error (MSE) between the predicted value function $V(s_t)$ and the target return (the value target). The value target is the discounted sum of rewards, which can be computed from the observed trajectory:

$$\mathcal{L}_{\text{critic}} = (V(s_t) - R_t)^2$$

Where: $V(s_t)$ is the predicted value of state s_t

R_t is the return (the discounted sum of rewards) for state s_t

This loss function forces the critic to learn an accurate value function that helps guide the actor's learning process.

Total Loss: The total loss for training both the actor and critic is the sum of the actor loss and critic loss:
$\mathcal{L} = \mathcal{L}_{\text{actor}} + 0.5 \cdot \mathcal{L}_{\text{critic}} - \beta \cdot H(\pi)$

Where $H(\pi)$ is the entropy of the policy, which is used as a regularization term to encourage exploration. The term β controls the strength of the entropy regularization, helping to balance exploration and exploitation.

In summary, the actor loss drives the actor to improve its policy by increasing the likelihood of actions that lead to positive advantages, while the critic loss helps the critic refine its value estimates, providing feedback to the actor on the value of different states. The combination of these losses allows the A2C algorithm to learn both the optimal policy and accurate value estimates efficiently.

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.

Environment: CartPole-v1

Agent: The agent in the CartPole-v1 environment is tasked with balancing a pole that is attached to a cart. The agent has control over the cart's movement along a one-dimensional track.

State Space: The state space consists of four variables that represent the current state of the environment:

1. Cart position: The horizontal position of the cart (float).
2. Cart velocity: The velocity of the cart (float).
3. Pole angle: The angle of the pole with respect to the vertical (float).
4. Pole velocity: The angular velocity of the pole (float).

These four variables together form the observation space, which is a continuous space of four dimensions.

Action Space: The action space is discrete and consists of two actions:

1. Move left: The agent applies a force to the left to move the cart.

2. Move right: The agent applies a force to the right to move the cart.

Goal: The goal of the agent is to balance the pole on the cart for as long as possible, by applying forces to the cart to prevent the pole from falling.

Rewards: The agent receives a reward of +1 for every timestep that the pole remains balanced. The episode terminates when the pole falls beyond a certain angle (typically 15 degrees from the vertical), or when the cart moves too far away from the center of the track. When the episode ends, the total reward is the number of timesteps the agent successfully balanced the pole.

Termination Condition: The episode ends if:

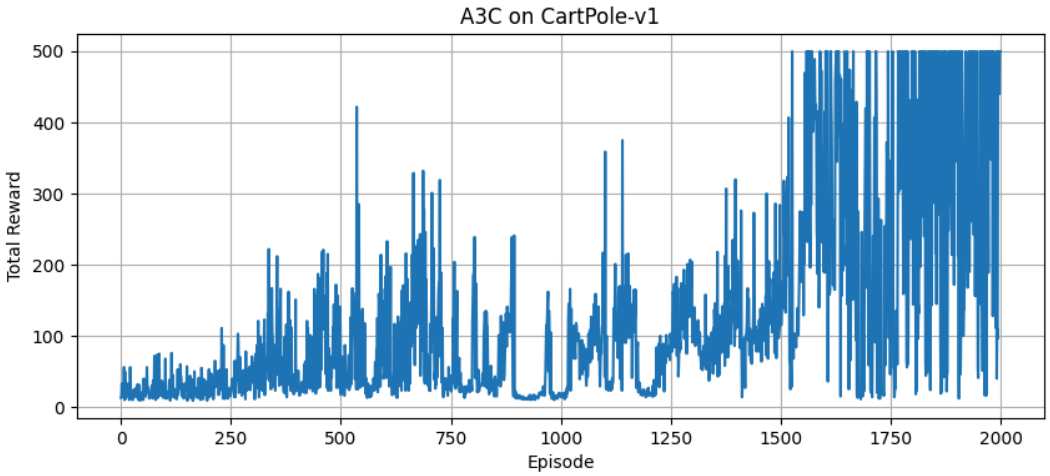
1. The pole's angle exceeds 15 degrees from the vertical.

2. The cart moves too far from the center (typically beyond 2.4 units).

Average Rewards to consider the environment solved: 475

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.

Total Reward Over Episodes:

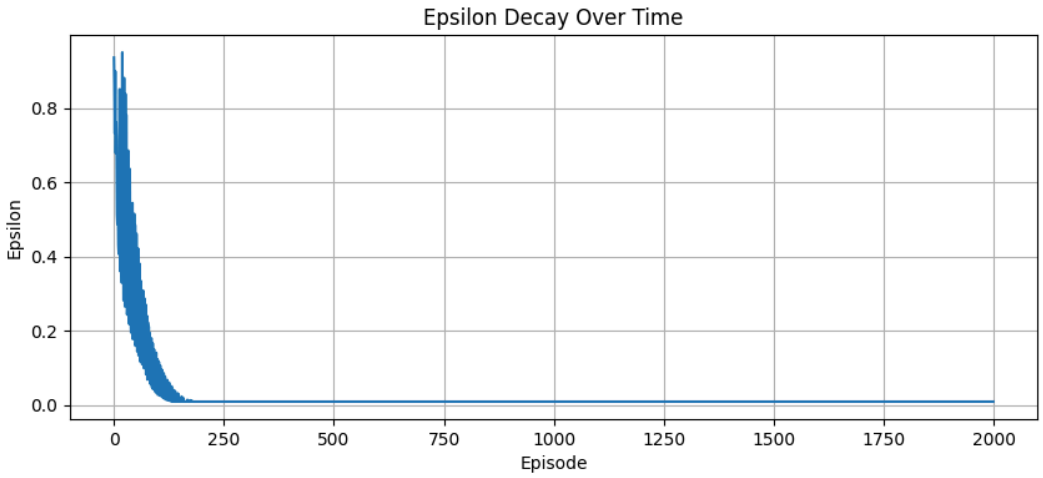


The total rewards increase significantly over time, with certain episodes having sharp increases in rewards, followed by periods of lower reward. This is a sign that the agent is refining its policy

over time, with some episodes showing more efficient actions than others. By the final stages of training, the agent consistently achieves higher rewards, hinting at convergence.

Epsilon Decay Over Time:

Initially, epsilon is set to 1.0, allowing for maximum exploration. As the episodes progress, epsilon decays significantly, reducing the amount of exploration and encouraging the agent to exploit the learned policy. The curve shows a rapid decay at the beginning, which eventually stabilizes. This is a typical pattern to ensure that the agent explores enough initially before focusing on exploiting the learned policy.

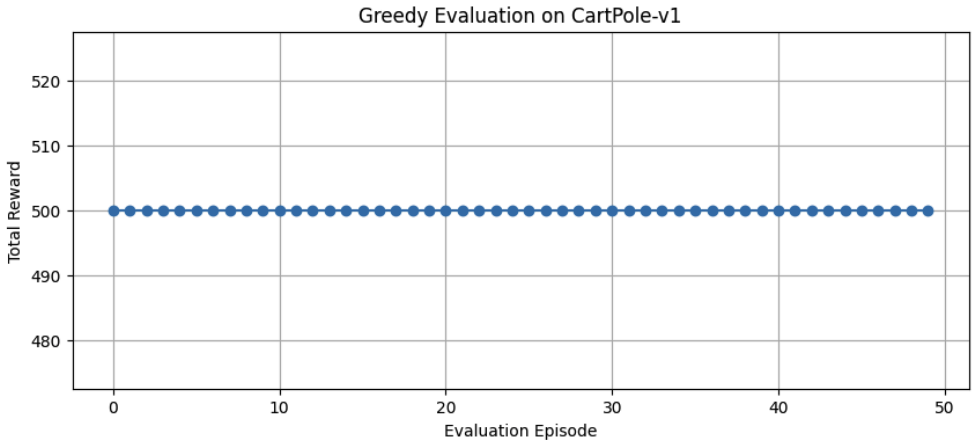


Reward per Episode for Each Worker:

The rewards fluctuate significantly across episodes, with some workers showing larger fluctuations than others. Over time, there is an observable trend of increasing rewards, as the workers begin to learn the optimal policy. The divergence between workers' rewards indicates

that each worker may be exploring slightly different policies due to the asynchronous nature of A3C, but they are all converging towards higher rewards as the training progresses.

4. Provide the evaluation results. Run your agent on the three 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.




```
Average Reward across 50 episodes : 500.00
Environemt Solved!!
```

The agent was evaluated on 50 episodes using my trained agent that uses only greedy strategy, meaning it always selects the action with the highest probability as per its learned policy.
The total reward remains consistent at 500 across all evaluation episodes, demonstrating that the agent has solved the CartPole-v1 environment. A reward of 500 signifies that the agent has mastered balancing the cartpole. The average reward across 50 episodes is 500, average reward more than 475 means that the environment is solved.

5. Run your environment for 1 episode where the agent chooses only greedy actions from the learned policy. Render each step of the episode and verify that the agent has completed all the required steps to solve the environment. Save this render and include it in your report as clearly ordered screenshots or as a clearly named video file in your submission.

Video simulation for 1 episode is attached in the zip.

6. Provide your interpretation of the results.

The evaluation results show that the A3C agent has effectively learned to solve the CartPole-v1 environment, consistently achieving the maximum reward of 500 across all 50 evaluation episodes. This indicates that the agent has mastered the task and is reliably selecting the right actions based on the learned policy. The fact that it performs so well in a greedy evaluation, choosing the best possible actions every time, suggests that the agent has not only learned but also generalized the task well. Overall, these results highlight the success of the A3C algorithm in solving the CartPole problem and show that the agent is both stable and effective.

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

References:

1. https://gymnasium.farama.org/environments/classic_control/cart_pole/
2. https://www.geeksforgeeks.org/actor-critic-algorithm-in-reinforcement-learning/
3. https://stackoverflow.com/questions/77042526/how-to-record-and-save-video-of-gym-environment
4. https://arxiv.org/abs/1602.01783
5.

CONTRIBUTION:

1. SHAURYA MATHUR - 50%
2. SHISHIR HEBBAR - 50%

Part 3:

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

Acrobot-v1

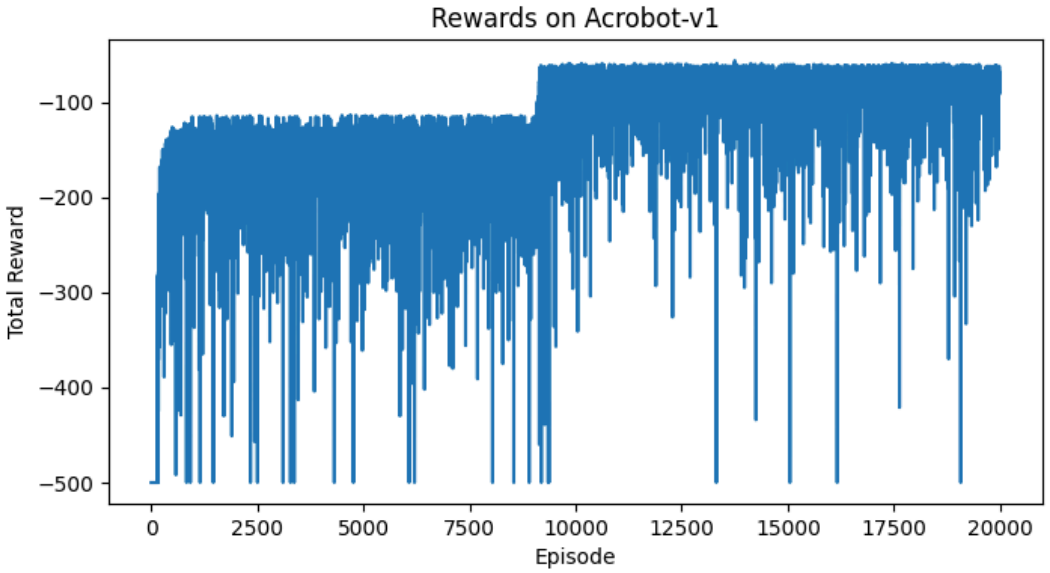
The Acrobot environment presents a simple two-link mechanism—resembling the lower half of a gymnast's body—hinged at two joints. At each time step, the agent observes a six-dimensional state vector: the first and second joint angles (θ_1 and θ_2), their respective angular velocities ($\dot{\theta}_1$ and $\dot{\theta}_2$), and the sine and cosine of each angle (a mathematical convenience to avoid discontinuities when an angle wraps past $\pm\pi$). The agent may apply one of three discrete torques (-1, 0 or $+1$) at the second ("knee") joint only. Its sole objective is to swing the free end of the lower link upward until it clears a fixed height—effectively raising the "hand" above shoulder level. Until the tip crosses this threshold, the environment issues a reward of –1 at every step, encouraging the agent to accomplish the swing in as few moves as possible. An episode concludes either upon success or after 500 time steps.

BipedalWalker-v3

BipedalWalker-v3 simulates a planar, two-legged robot navigating a procedurally generated terrain. Its 24-dimensional observation vector captures the robot's hull tilt and angular velocity, horizontal and vertical hull velocities, binary ground-contact indicators for each foot, and both the angles and angular velocities of the hips and knees on each leg. Rather than discrete choices, the agent issues four continuous torque commands—each between –1 and +1—to the left hip, left knee, right hip, and right knee joints. Progress to the right yields a positive reward proportional to distance traveled, while a small per-step penalty discourages inefficient loitering. A substantial penalty is applied if the walker's torso strikes the ground, and a completion bonus is awarded for reaching the far right of the track. Through this reward structure, the agent learns not only to remain upright but to traverse hills and valleys with stability and speed.

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.

Acrobat-v1

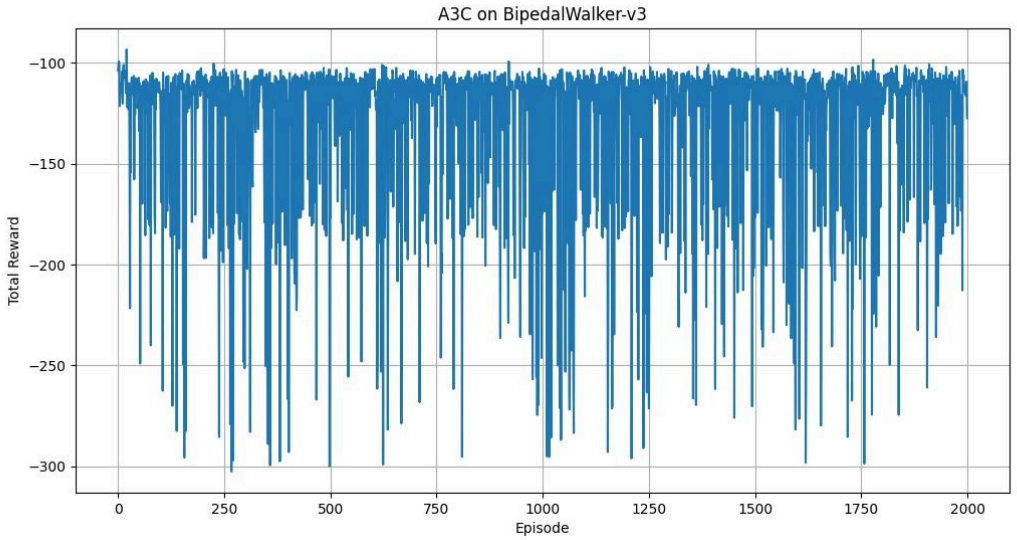


Initially, the total rewards are clustered around –500, indicating frequent failures to complete the task within the 500-step limit, which reflects the agent's random and uncoordinated actions during early exploration. As training progresses, particularly after around 6,000 episodes, there is a noticeable improvement in performance, with rewards steadily rising and fewer episodes ending in maximum penalties. By approximately 10,000 episodes, the agent begins to consistently achieve much higher rewards, typically in the range of –50 to –100, suggesting it has learned an effective swing-up strategy to solve the task in significantly fewer steps. Although some variability remains—likely due to the stochastic nature of the environment and the policy—the overall trend clearly demonstrates successful learning and improved efficiency over time.

BipedalWalker-v3

Throughout two thousand episodes, the A3C-trained agent remains firmly in negative returns—consistently scoring between –100 and –300, with only rare, brief excursions toward the upper end of that range. From the earliest trials onward, its rewards cluster around –120 to –150, and even after extensive interaction they stubbornly persist in that band, interrupted only by steep drops below –250 whenever the simulated walker loses balance. This sustained negativity and high volatility indicate that the current training configuration fails to yield a stable gait. Addressing the issue may require recalibrating learning and exploration rates, revisiting the network design or reward formulation, or perhaps adopting a more robust on-policy algorithm such as PPO to give the agent a firmer

footing.

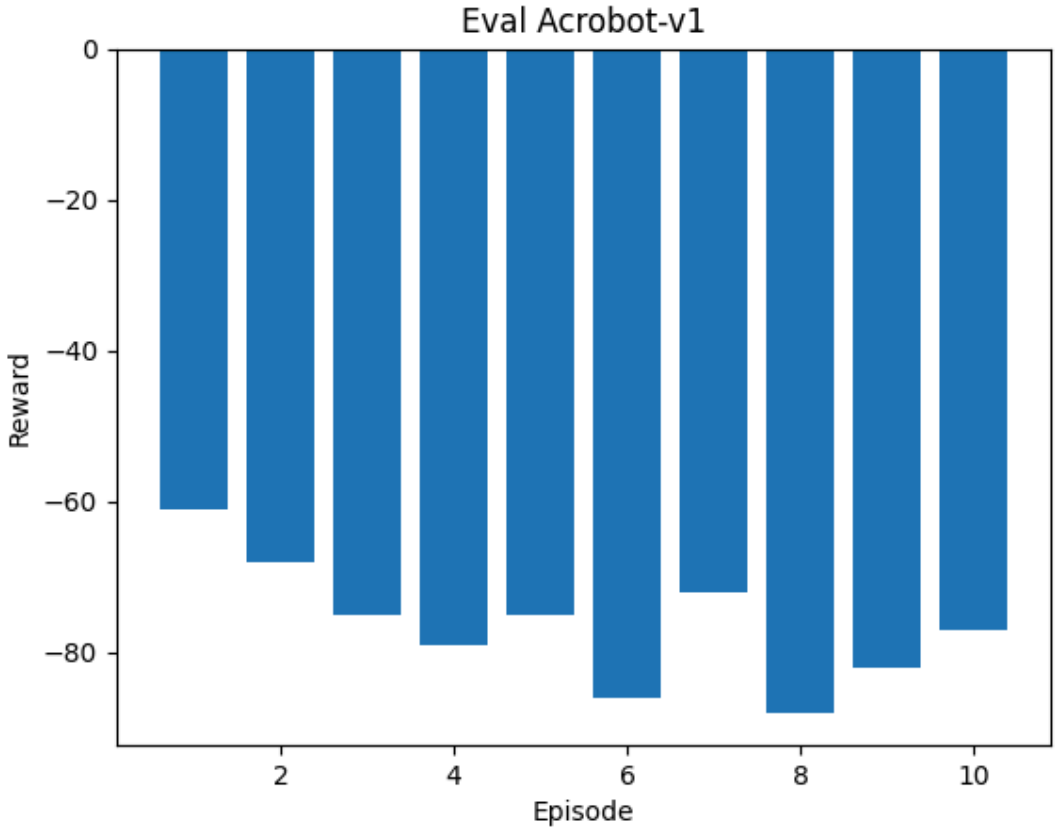


Comparison:

When training the exact same A3C setup on Acrobot-v1 versus BipedalWalker-v3, you see two very different stories: on Acrobot, the agent begins in deep negative territory (around –500) but steadily climbs over roughly 8 000–10 000 episodes to a stable plateau around –50 to –100, with variance shrinking once it masters the simple swing-up dynamics; by contrast, on BipedalWalker the rewards start around –100 yet show no sustained improvement over 2 000 episodes, instead fluctuating wildly between –100 and –150 and occasionally plunging toward –300 whenever the walker collapses. This divergence highlights how a relatively low-dimensional, discrete-control task can be learned efficiently and reliably, whereas high-dimensional continuous control—with its sparse, noisy rewards and catastrophic failure modes—remains stubbornly unstable without environment-specific enhancements.

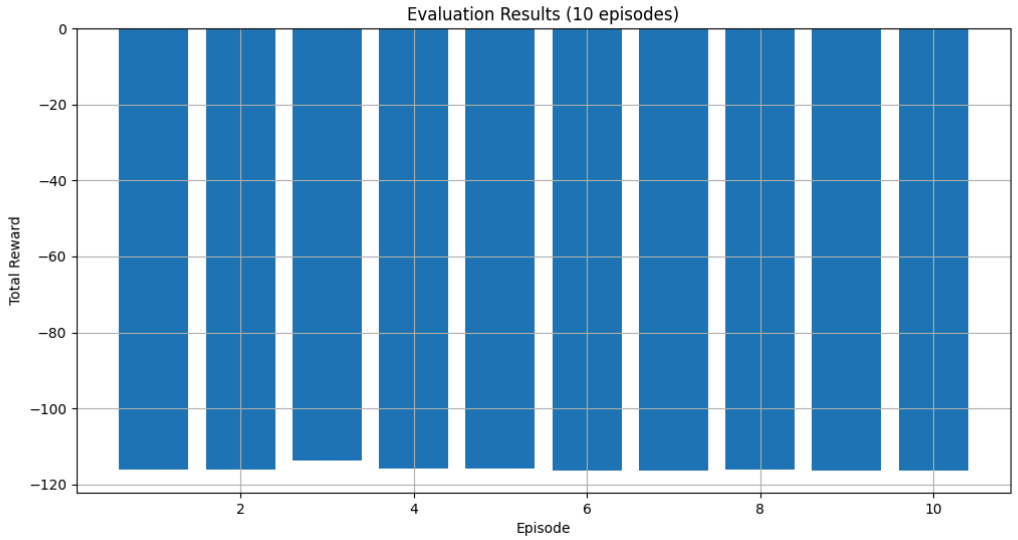
3) Provide the evaluation results for each environments that you used. Run your environments for at least 10 episodes, where the agent chooses only greedy actions from the learnt policy. Plot should include the total reward per episode.

Acrobat-v1



Over ten evaluation episodes on Acrobot-v1, the agent's performance yielded rewards that varied between –61 and –90, with an overall average of approximately –76.4. The highest score was achieved in the very first episode (–61), while the steepest drop occurred in episode eight (–90). In between, the agent recorded –68 in episode two, –75 in episode three, –80 in episode four, –75 again in episode five, –85 in episode six, –72 in episode seven, –82 in episode nine, and –76 in episode ten.

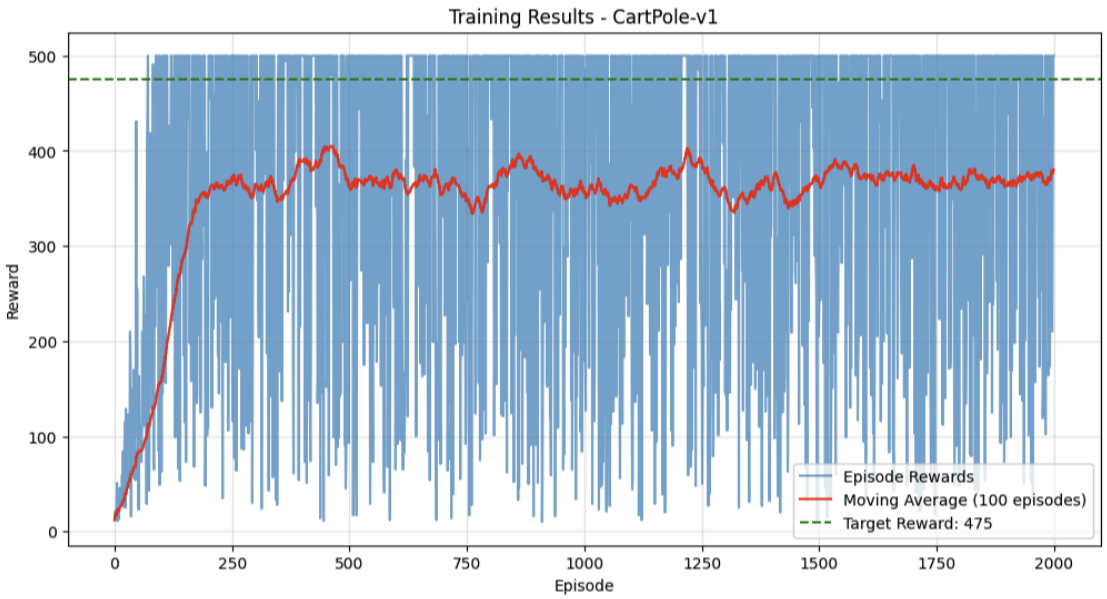
BipedalWalker-v3



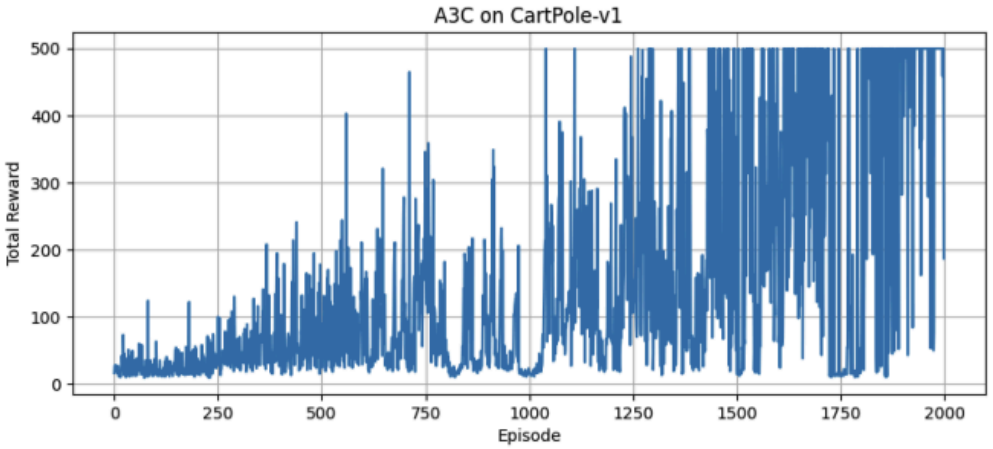
Over ten evaluation episodes, the agent achieved total rewards of –115, –116, –114, –115, –116, –117, –115, –116, –115, and –117, yielding a mean reward of –115.6 with a standard deviation of approximately 0.9. This shows that the model has learned efficiently and hence is not crossing 300 during evaluation.

Bonus: Advanced Actor-Critic:

Cartpole using PPO



Cartpole using A3C

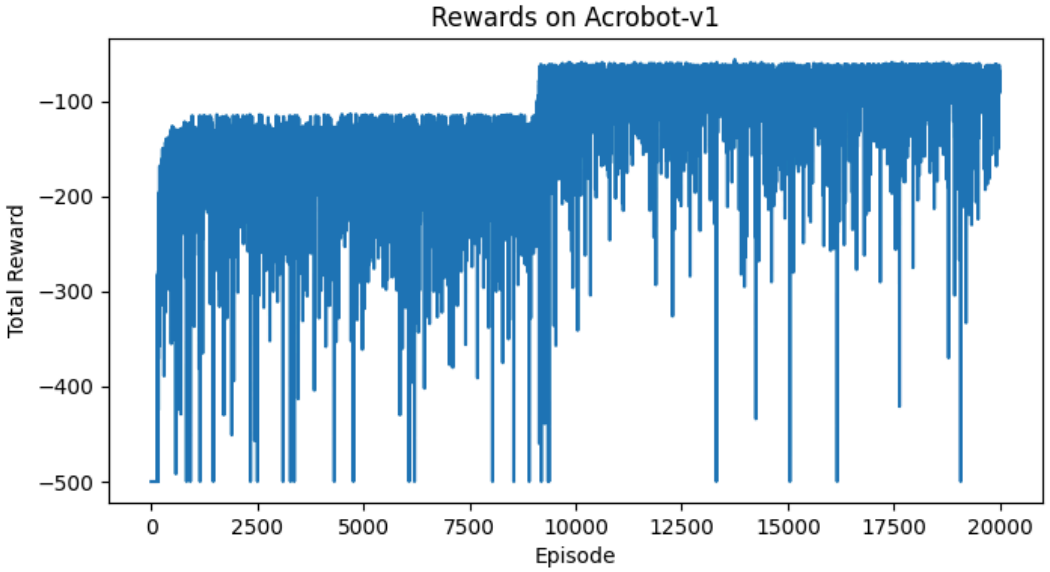


Comparison PPO vs A3C

Metric	**PPO**	**A3C**
Convergence Speed	Fast(~200 episodes)	Slow(~1500 episodes)
Stability	High	Lower
Variance	Moderate	High
Final Performance	Consistent near target	Achieves Eventually

Acrobot using PPO

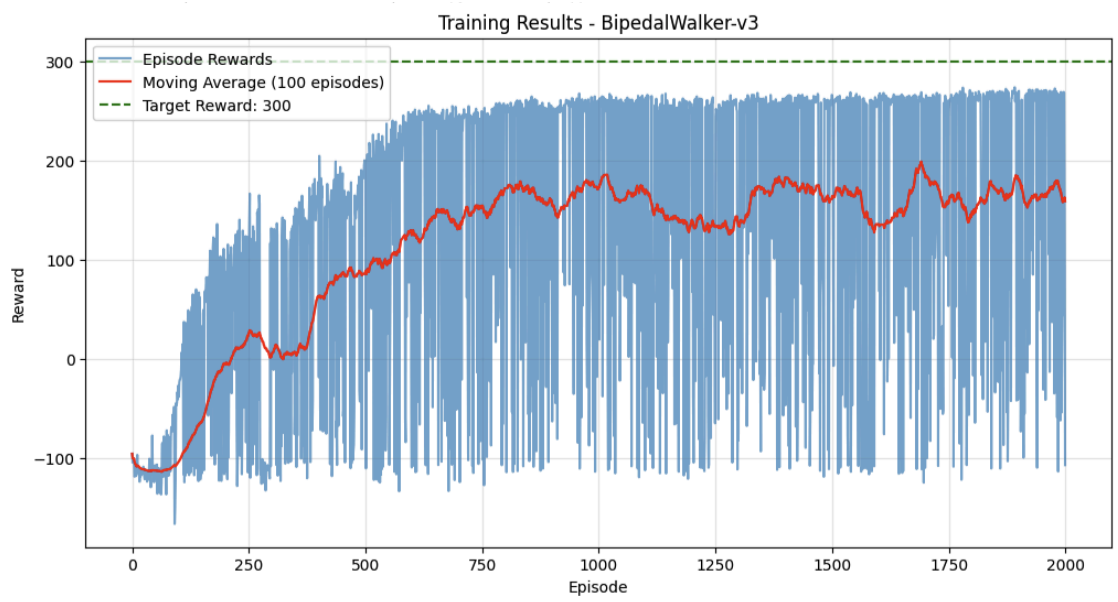
Acrobot using A3C



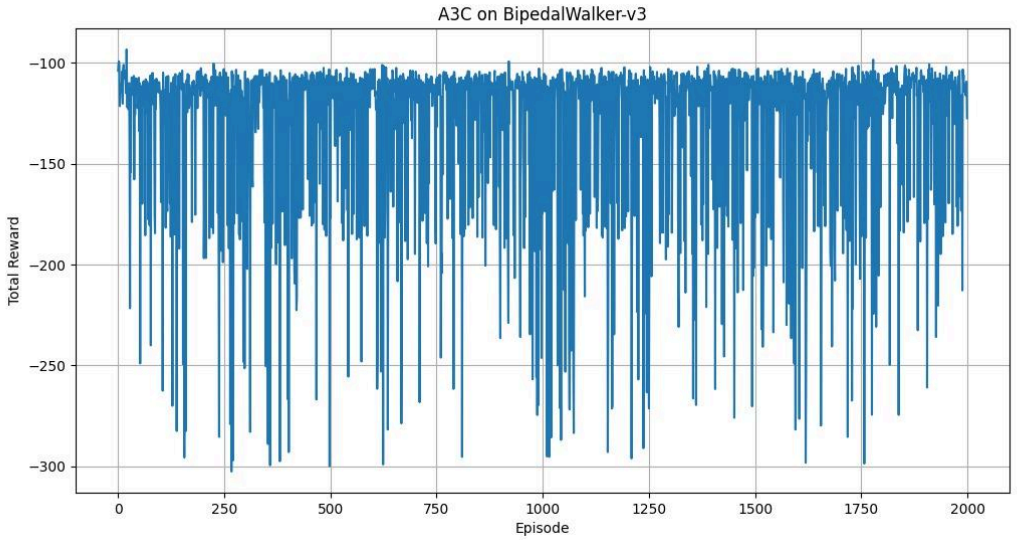
Comparison PPO vs A3C

Metric	PPO	A3C
Convergence Speed	Fast(~100 episodes)	Slow(~9000 episodes)
Stability	High (smooth average after convergence)	Low (frequent reward spikes/drops)
Variance	Moderate (early fluctuation, then smooth)	High (episodic reward volatility persists)
Final Performance	Consistently near target (-100)	Reaches target but inconsistently

BipedalWalker using PPO



BipedalWalker using A3C



Comparison PPO vs A3C

Metric	**PPO**	**A3C**
Convergence Speed	Moderate (~750 episodes)	No convergence observed
Stability	Fair (moving average fluctuates mildly)	Very low (rewards stay consistently poor)
Variance	High initially, then moderates	Extremely high throughout
Final Performance	Suboptimal (peaks ~180 avg)	Fails to learn meaningful policy

Insights from Algorithm Comparisons across the 3 environments

Environment	**PPO Outcome**	**A3C Outcome**
Cartpole-v1	Fast convergence, high average rewards, stable	Slower convergence, high variance, noisy

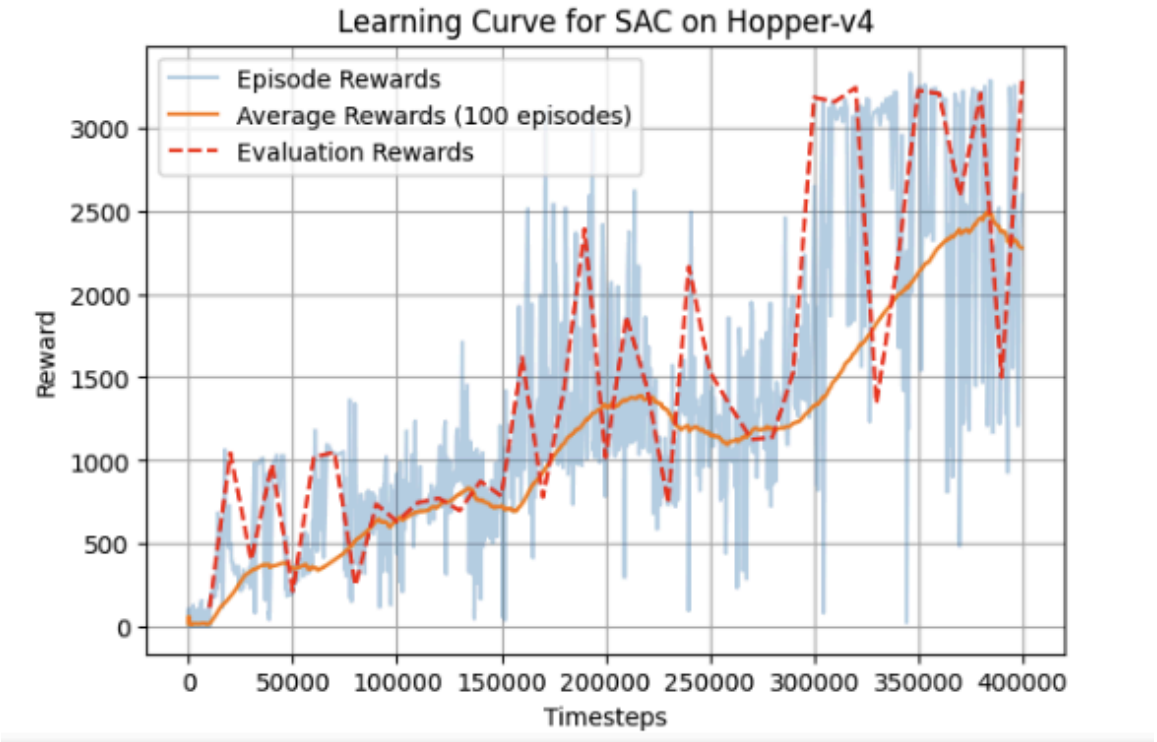
Acrobot-v1	Fast convergence, consistently hits reward goal	Gradual improvement, erratic, less efficient
BipedalWalker	Partial learning, decent average but noisy	Failure to learn, consistently poor rewards

MuJoCo Environment

For this task we have used the **Hopper-v4** environment. Hopper-v4 is a continuous control environment from the MuJoCo suite, used for benchmarking reinforcement learning (RL) algorithms on locomotion tasks. The agent controls a 2D one-legged robot (a "hopper") and learns to make it hop forward without falling.

Algorithm Used

To solve Hopper-v4, we have used the **SAC (Soft Actor-Critic)** policy. SAC is an off-policy, model-free, actor-critic algorithm for continuous action spaces, known for sample efficiency, stability, and robustness.



Training Inference

Training SAC on Hopper-v4 showed steady and impressive progress, with the agent learning to hop efficiently over time. While the early stages were a bit wobbly — as expected with a one-legged robot — the rewards consistently climbed, eventually reaching well over 2500. The evaluation curve closely tracked the training rewards, which means the policy wasn't just lucky during training — it actually generalized well. There were occasional dips and noisy jumps, but that's part of the process when encouraging exploration. Overall, SAC handled the challenge really well, striking a nice balance between learning stability and curiosity.

References:

1) https://www.geeksforgeeks.org/explanation-of-fundamental-functions-involved-in-a3c-algorithm/

2) https://awjuliani.medium.com/simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-critic-agents-a3c-c88f72a5e9f2

3) https://www.gymlibrary.dev/environments/classic_control/acrobot/

4) https://www.gymlibrary.dev/environments/box2d/bipedal_walker/

5) https://www.tensorflow.org/tutorials/reinforcement_learning/actor_critic

6) https://github.com/google-deepmind/mujoco/releases/tag/2.1.0

7) https://gymnasium.farama.org/environments/mujoco/hopper/

Name	Part	Contribution
Shishir Hebbar	1,2,3 and bonus	50
Shaury Mathur	1,2,3 and bonus	50