1. **Discuss the A2C/A3C algorithm you implemented.**

I implemented the A3C algorithm to train an agent in the CartPole-v1 environment. The algorithm uses two neural networks: the Actor, which predicts action probabilities for the agent, and the Critic, which estimates the value of the current state. I implemented both global (shared) and local versions of these networks, allowing multiple threads to train simultaneously. Each thread independently interacts with the environment, periodically syncing its local networks with the global ones. During training, each thread explores the environment, collects rewards, and computes discounted returns. Using these returns, the thread calculates two types of losses: policy loss for updating the Actor and value loss for updating the Critic. These losses are used to backpropagate gradients, which are applied to the global networks for synchronization. After training, I evaluated the global Actor's performance by running multiple test episodes, selecting the best actions at each step, and calculating the rewards.

1. Flow of the algorithm:
    - Initialization:
        - Global networks (actor and critic) and optimizers are created and shared among threads.
    - Worker:
        - Interacts with the environment, collecting rewards, actions, and states.
        - Computes discounted rewards (future cumulative rewards).
        - Calculates policy loss (actor update) using the advantage and value loss (critic update) using the difference between predicted and actual discounted rewards.
        - Gradients from the local network are applied to the global network.
    - Evaluation:
        - After training, the global actor network is evaluated in a greedy manner to determine its performance.
2. Key Features:
    - Parallelism: Multiple threads speed up training by sampling diverse experiences.
    - Stability: Normalizing rewards and using smooth L1 loss helps stabilize training.
    - Efficiency: Sharing parameters avoids redundant computations.

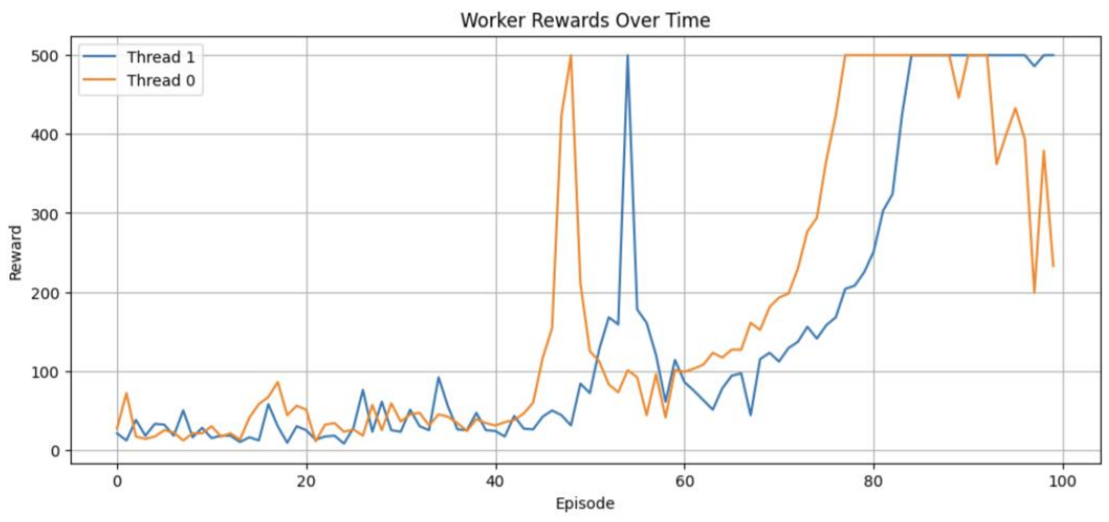
2. What is the main difference between the actor-critic and value based approximation algorithms?

The main difference between actor-critic and value-based algorithms lies in how they represent and optimize the policy. Value-based methods indirectly derive the policy by learning a value function and selecting actions greedily based on the highest value, which works well for discrete action spaces but struggles in continuous domains. On the other hand, the actor-critic method explicitly models the policy (actor) and optimizes it directly while using a separate value function (critic) to estimate state values and guide policy updates. In comparison to value-based methods the Actor-critic methods are more suitable for continuous or high-dimensional action spaces, but they require training two networks, making them more complex. Value-based approaches are simpler and computationally efficient for discrete problems, whereas actor-critic excels in environments where sampling actions directly from a policy is more beneficial.

Describing Cartpole-v1 Environment

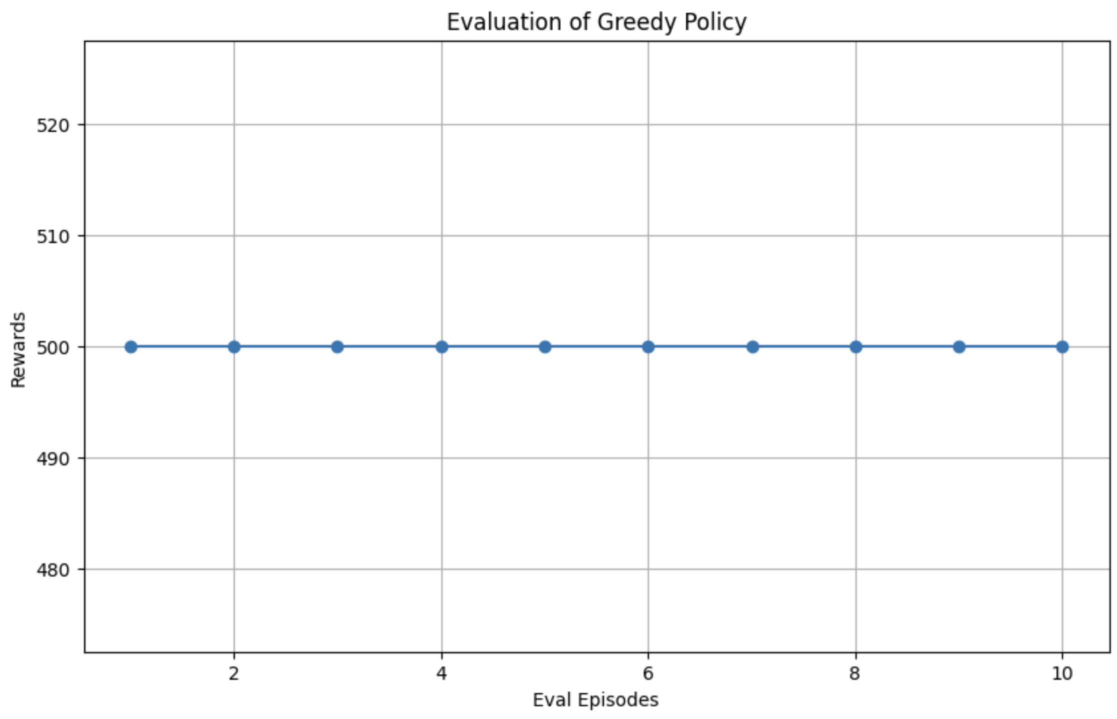
- State Space: A 4-dimensional vector representing the cart position, cart velocity, pole angle, and pole angular velocity.
- Action Space: Two discrete actions—move the cart left or right.
- Goal: Keep the pole upright and the cart within the track boundaries for as long as possible.
- Rewards: The agent receives a reward of +1 for every time step the pole remains balanced.
- Termination: The episode ends if the pole angle exceeds a threshold, the cart moves out of bounds, or a maximum time step is reached.

Training reward over episodes



The graph illustrates the rewards earned by two worker threads over episodes in the CartPole environment. Initially, both threads show low and inconsistent rewards due to random exploration. Over time, rewards increase as the workers learn better policies. Around episode 60, both threads achieve higher stability, with rewards consistently approaching the maximum possible value (500). However, thread-specific fluctuations indicate varying learning speeds, potentially due to differences in experience sampling or policy updates. Overall, the workers demonstrate successful learning by achieving the optimal rewards.

Evaluating agent on greedy policy



The graph illustrates the performance of the trained policy during evaluation episodes. Rewards are consistently at 500, the maximum reward attainable for this environment, proving that the agent has learned the optimal policy.

References:

https://app.datacamp.com/learn/courses/deep-reinforcement-learning-in-python

https://www.gymlibrary.dev/

Code for graphs and description of the environment was generated using ChatGPT