Assignment 3 - Actor-Critic

Anurima Vaishnavi Kumar

April 29, 2023

1 Discuss the Q Actor-critic algorithm you implemented.

The Q Actor-Critic (QAC) algorithm is a type of reinforcement learning algorithm that combines ideas from both Q-learning and actor-critic methods. QAC aims to learn an optimal policy by estimating the Q-values (action values) and using them to update both a policy (the actor) and a value function (the critic).

Actor:

- In a reinforcement learning algorithm like Q Actor-Critic (QAC), the actor is responsible for selecting actions based on the current policy. The policy is a function that maps states to a probability distribution over possible actions. The role of the actor is to learn an optimal policy that maximizes the expected reward over time.

  In QAC, the actor updates the policy using the Q-value function as a critic. The Q-value function estimates the expected future reward of taking a particular action in a given state. By updating the policy to favor actions with higher Q-values, the actor can gradually learn to select actions that lead to higher overall reward.

  The actor plays a crucial role in QAC because it determines the actions taken by the agent in each time step. If the actor selects actions that lead to low reward, the agent will perform poorly and learn a suboptimal policy. However, if the actor can learn to select actions that lead to high reward, the agent can learn an optimal policy that maximizes the long-term reward.

Critic:

- n a reinforcement learning algorithm like Q Actor-Critic (QAC), the critic is responsible for estimating the expected future reward of taking a particular action in a given state. The role of the critic is to learn a Q-value function that maps states and actions to the expected future reward.

  The Q-value function estimates the value of taking a particular action in a given state, and it is used by the actor to update the policy. Specifically,

the actor updates the policy to favor actions with higher Q-values, which can lead to higher overall reward.

The critic plays a crucial role in QAC because it provides a measure of the expected future reward of taking an action in a given state. This allows the agent to evaluate different actions and choose the one that leads to the highest expected reward.

Furthermore, the critic can be used to estimate the state value function, which is the expected future reward starting from a given state. This can be useful for tasks where the goal is to maximize the long-term reward, as it allows the agent to estimate the value of each state and take actions that lead to higher overall reward.

Training process:

1. You reset the environment each time an episode ends. You run the environment until it reaches termination or it achieves the maximum reward attainable for that environment

2. Initially the agent selects actions randomly until a certain point. If the epsilon value is value is greater than the random value generated, we take a random action among all possible actions. If the agent has not taken enough time steps, then also we take a random action

3. When the agent doesn't satisfy any of the above two conditions, the agent now takes an action sampled from the actor policy.

4. I kept the learning rate of the actor to be lower in comparison to the critic since I wanted the critic to learn faster since the actor depends on critic to give out optimal actions

5. Once there is enough experience instances in the replay buffer we start the training of the networks. We first start the training of the critic and then the actor since you want the critic to stabilize well for the actor to learn.

6. I have used two networks for both actor and critic and performed soft updates on the target networks whenever the gradients are calculated and correspondingly updated. I follow the approach of DQN where you have one target which is moving and the other one learning from the stablized target.

7. I have used Adam optimizer for both the networks.

8. Instead of using value function as the output for the critic network, we use the Q values as the output function here.

9. The use of replay buffers, soft updates, learning rate of critic greater than actor, being greedy in initially have been found to be useful techinques in faster convergence

Convergence:

- Convergence value for all the environments have been mentioned in the code. Convergence happens when the agent achieves maximum reward. Training continues until the agent goes to the point of convergence. After the agent reaches the point, you can stop the training. Convergence can be determined in many ways. I used total rewards per episode graph to see whether or not it was converging. Moving average over 100 episodes is also another good metric

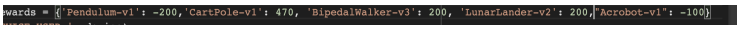

Figure 1: Maximum rewards

The Q Actor-Critic (QAC) algorithm combines the advantages of value-based and policy-based methods in reinforcement learning. The critic network estimates the values of actions in a given state, allowing the agent to efficiently explore the environment. Meanwhile, the actor network learns a stochastic policy that directly maps states to actions, enabling the agent to take actions that lead to higher reward.

By combining these two approaches, QAC achieves a balance between exploration and exploitation, and is able to learn an optimal policy more efficiently than either approach alone. The critic network provides guidance on which actions are likely to lead to higher reward, while the actor network explores the environment and learns a policy that maximizes the long-term reward.

Overall, the QAC algorithm is a powerful and flexible method for solving reinforcement learning problems, and has been successfully applied to a wide range of tasks in areas such as robotics, game playing, and natural language processing **Important components of the implementation:**

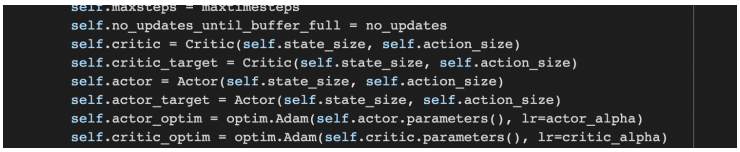

Figure 2: Actor and Critic Networks

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

Actor-critic and value-based approximation algorithms in reinforcement learning differ in their approach to learning an optimal policy. **V**alue-based algorithms, such as Q-learning and SARSA, learn a value function that estimates

the expected future reward for each state-action pair. The agent selects the action with the highest expected future reward in each state, indirectly learning a policy. **I**n contrast, actor-critic algorithms learn both a value function and a policy function. The critic estimates the value of taking a particular action in a given state, and the actor directly learns a policy that maps states to actions. The critic provides guidance to the actor by estimating the value of different actions, allowing the actor to update the policy accordingly. **C**hoosing between actor-critic and value-based approximation algorithms depends on the specific problem being solved and the available resources for training. The algorithms offer different trade-offs in terms of sample efficiency, complexity, and stability, which must be considered in selecting the most appropriate approach. **V**alue-based algorithms balance exploration and exploitation using an -greedy policy, while actor-critic algorithms use a stochastic policy that directly maps states to actions, allowing for more flexible exploration and exploitation. **V**alue-based algorithms are off-policy, meaning they learn from a different policy than the one they are following, which can lead to instability. Actor-critic algorithms are on-policy, meaning they learn from the same policy they follow, leading to more stable learning but requiring more exploration. **A**ctor-critic algorithms are generally more sample-efficient because they learn both a value and policy function. They can require careful tuning of hyperparameters, making them more complex to implement than value-based algorithms, which only involve learning a value function.

3 Briefly describe THREE environments that you used (e.g. possible actions, states, agent, goal, rewards, etc)

LunarLander-v2:

- **State**: The state of the environment is an eight-dimensional vector consisting of the position and velocity of the lander, the angle and angular velocity, and whether the legs are in contact with the ground.
- **Action**: The agent can take one of four possible actions: do nothing, fire the left orientation engine, fire the right orientation engine, or fire the main engine.
- **Reward**: The reward is determined by the proximity of the lander to a designated landing pad, with a larger reward given for landing closer to the center of the pad. Negative rewards are given for using engine thrust and for crashing.
- **Goal**: The goal is to successfully land the lunar lander on the designated landing pad without crashing, using the least amount of fuel possible.

'CartPole-v1'

Rewards: Each time step the agent successfully balances the pole yields a reward of +1. The episode ends when the pole falls over or the agent balances it for a predetermined number of time steps. **Actions**: The agent can take one of two possible actions: move the cart left or right. **State space**: The state of the environment is defined by the position and velocity of the cart and pole. Specifically, the state is a four-dimensional vector consisting of the position and velocity of the cart, and the angle and angular velocity of the pole. **Action space**: The action space is discrete and consists of two possible actions: move the cart left or right. **Goal state**: The goal is to balance the pole for as long as possible without it falling over.

Acrobot-v1:

- **State**: The state of the environment is defined by the position and velocity of two joints in a two-link pendulum system. Specifically, the state is a six-dimensional vector consisting of the cosine and sine of the two joint angles and their corresponding angular velocities.
- **Action**: The agent can take one of three possible actions: apply torque to the first joint in the clockwise direction, apply torque to the first joint in the counterclockwise direction, or apply torque to the second joint in the clockwise direction.
- **Reward**: The reward is -1 for each time step until the goal state is reached. The goal state is defined as having the two links touch the ground, which requires the pendulum to swing up and rotate over the top.
- **Goal**: The goal is to get the two links of the pendulum to touch the ground, which requires the pendulum to swing up and rotate over the top.

4 Show and discuss your results after training your Actor-Critic agent on each environment.

4.1 CartPole Environment

- **Training reward per episode**: You can see that the graph is consistently growing. The graph got converged around 900 episodes and it achieved a reward of 500 which means the environment has been solved
- **Factors contributing to stability**:
    1. *Initiliazation values*: The actor and critic learning rates have played a huge role. The critic has a faster learning rate in comparison to actor which is essential for the critic to satbilize initially and for the actor to learn. I planned on changing the learning rates later on making the actor learn faster than critic after certain period of time

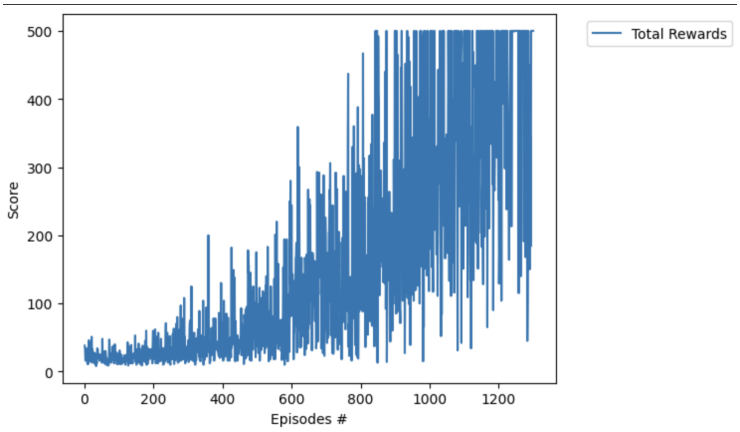

Figure 3: This graph shows the training reward per episode for the CartPole environment, using the Q Actor-Critic algorithm. The algorithm was able to learn to balance the pole for longer periods of time, as evidenced by the increase in average reward over the course of training. The rewards fluctuated somewhat during training, but generally trended upwards. Overall, the Q Actor-Critic algorithm proved to be effective in learning an optimal policy for the CartPole environment.

2. *Choosing actions* Being random about selecting actions in the beginning made the most sense to me. I didnt disturb the actor network until the replay buffer was full to certain point and the learnt something. To ensure some randomness in selections of action later during the training, I used epsilon greedy function for selection of actions
3. *Soft Updates* By using soft updates, QAC is able to maintain a stable target network, which in turn leads to more stable and efficient learning of the optimal policy. I think this played an important role in ensuring the actor and critic learns from a stable target actor and target critic networks

4.2 Acrobot Environment

Factors contributing to stability:

(a) **Experience replay:** I used experience replay to store past experiences in a buffer and randomly sample them during training. This helps to break the temporal correlations in the data, leading to more stable learning. Experience replay was found to be really useful here

(b) **Target network:** Using a separate target network that is updated periodically to reduce the variance in the value estimates. This lead to a more stable learning process and better generalization to new

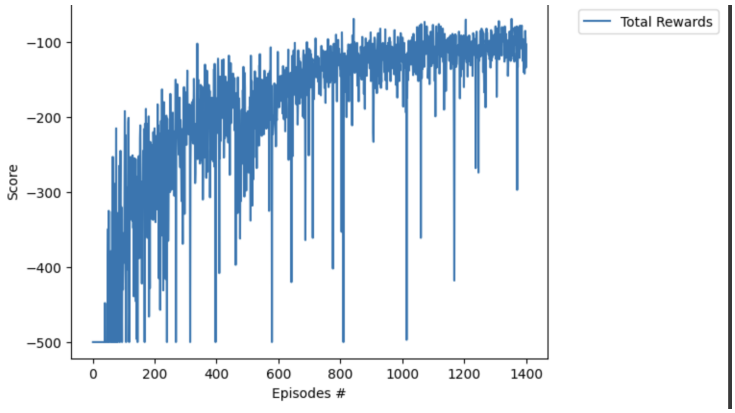

Figure 4: This graph depicts the training reward per episode for the Acrobot environment using the Q Actor-Critic (QAC) algorithm. The algorithm was able to learn to swing up the pendulum and achieve the goal state of touching the ground, as evidenced by the increase in average reward over the course of training. The rewards varied during training, but the trend was generally positive. Overall, the QAC algorithm was effective in learning an optimal policy for the Acrobot environment, leading to improved performance of the agent.

states. Additionally, using the two dependent networks was found to be extremely useful for this environment

(c) **Exploration** Sampling actions from the probability distribution of the actor network is a useful technique to improve policy discovery, particularly in environments with large or continuous action spaces. This approach allows the agent to explore a wider range of actions and learn a more comprehensive policy. By considering the entire distribution of possible actions, the agent can better understand the trade-offs between different actions and make more informed decisions. This technique is particularly valuable in situations where the action space is large or continuous, as it allows for more flexible and expressive policies. Acrobot has large continuous action space hence QAC seemed to have performed really well

The agent trains on Acrobot-v1 environment for 3500 episodes with a convergence threshold of -100, using learning rates of 0.0005 for critic and 0.00001 for actor. The replay memory buffer size being 3000. The maximum timesteps for each episode being 100000. The discount factor being 0.99. The hidden dimensions used for all the 4 networks being 128. I used three layers for all the actor and critic networks. The environment started converging at around episode 800. The agent has achieved the threshold reward -100 by the episode number 1400

4.3 Lunar Lander Environment

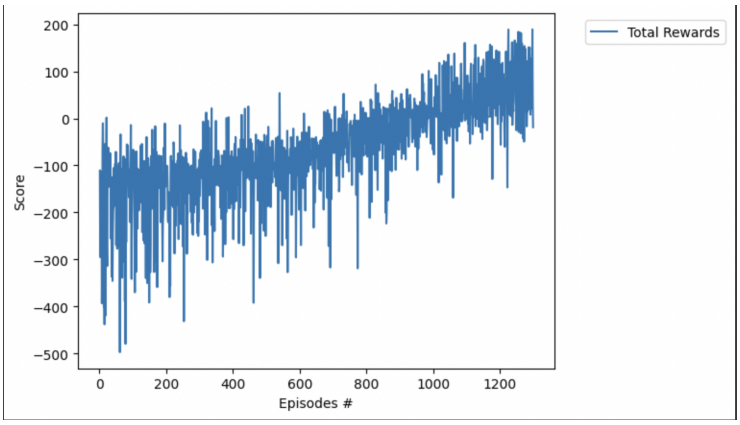

Figure 5: he provided graph displays the training reward per episode for the Lunar Lander environment, trained using the Q Actor-Critic algorithm. The algorithm was successful in learning to safely land the spacecraft, as demonstrated by the increase in average reward over the course of training. Although the rewards fluctuated somewhat during training, there was a general upward trend, indicating that the algorithm was effectively learning an optimal policy. These results suggest that the Q Actor-Critic algorithm is a promising approach for solving the Lunar Lander environment.

- **Training reward per episode**: You can see that the graph is consistently growing. The rewrad value kept increasing from the beginning and it reached the threshold value of 200 in between episodes 1000 and 1200
- **Factors contributing to stability**:
  (a) **Continuous action space:** The Lunar Lander environment has a continuous action space that is difficult to explore efficiently. QAC's actor-critic architecture and the ability to sample actions from the actor network's probability distribution allow for more effective exploration.
  (b) **Complex dynamics:** The Lunar Lander environment has complex physics and dynamics, making spacecraft control challenging. QAC's ability to learn a value function and a policy function simultaneously can help it better understand the complex dynamics of the environment, leading to more stable learning.
  (c) **Experience replay:** Experience replay is particularly helpful in the Lunar Lander environment, allowing the agent to learn from past experiences and avoid catastrophic events like crashing the spacecraft. By breaking the correlation between consec-

utive samples, experience replay can help to reduce the impact of outliers and improve stability.

5 Evaluation results for each environment

5.1 Cartpole-v1 Environment

The agent has achieved the maximum attainable reward of 500 during the evaluation for all the 10 episodes. This means that the agent has converged completely since the rewards remained constant during the evaluation

Episode	Total Reward
1	500.0
2	500.0
3	500.0
4	500.0
5	500.0
6	500.0
7	500.0
8	500.0
9	500.0
10	500.0

Figure 6: Evaluation results for CartPole-v1 environment.

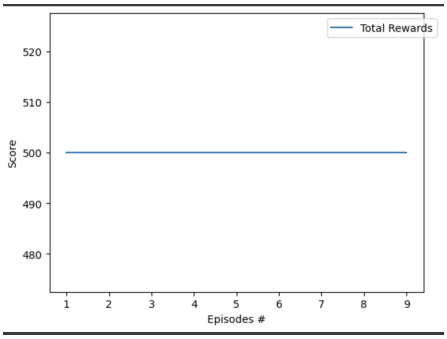

Figure 7: Evaluation Result

5.2 LunarLander-v2 Environment

The agent has not fully converged yet even though you see the convergence happening during the total rewards per episode graph. Agent is still receiving tremendously low rewards during some episodes. But you can see that the agent has clearly learned since the total rewards per episode graphs during training show that the agent received even more lower rewards ranging between -500 to 0 initially.

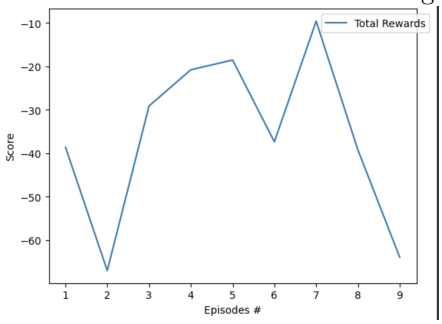

Figure 8: Evaluation Results

The agent has not fully converged yet even though you see the convergence happening during the total rewards per episode graph. There is no constant reward during the evaluation. The agent is receiving lesser rewards than the threshold value of -100 for convergence for a few episodes. You can see that the agent has learned since the range of episodes during evaluation is [-120,-80]

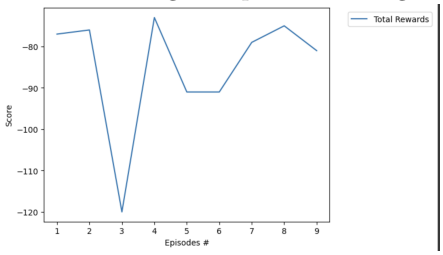

Figure 9: Eval Results for Acrobot-v1 Environment