CSE 546 Assignment 3 - Actor-Critic Algorithms

Sailesh Reddy
Department of Computer Science
University at Buffalo, SUNY
Buffalo, NY 14260
saileshr@buffalo.edu

Mohammed Nasheed Yasin
Department of Linguistics
University at Buffalo, SUNY
Buffalo, NY 14260
m44@buffalo.edu

Abstract

This report presents our experiments on three environments, `CartPole-v1`, `LunarLander-v2` and `InvertedPendulum-v4`. These environments were selected from the `ClassicalControl`, `Box2D` and `MuJoCo` collections respectively in the Gymnasium library [2]. We applied Q Actor-Critic Algorithm to solve these environments and conducted a case study on the outcomes.

1 Q Actor-Critic Algorithm

1.1 Network Architecture

The policy(actor) network takes the state as input and outputs probability distributions over the action space. In the case of discrete action space, the value function (critic) net again takes the state as input and outputs estimate the Q value for each action In continuous action space. In the case of continuous action space, however, the value function (critic) net takes the state-action pair as input and outputs the Q value for that particular state-action pair.

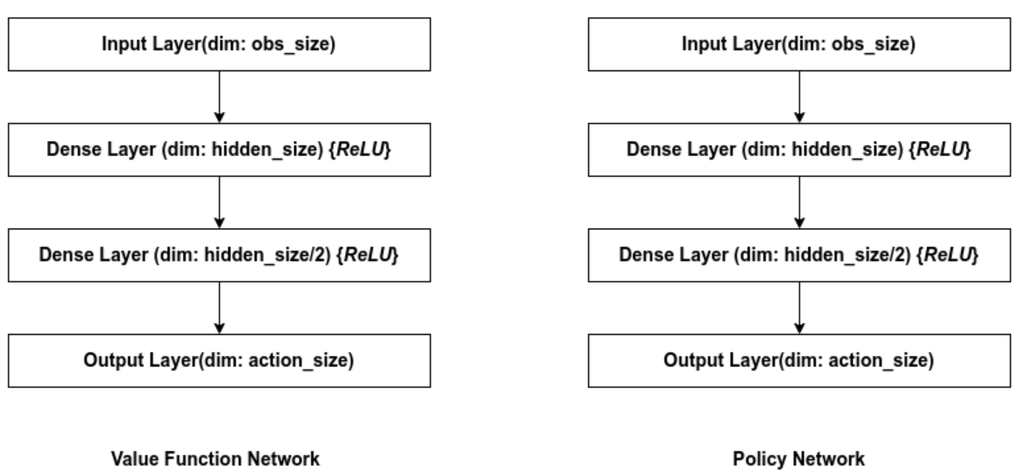

Figure 1: Networks for Discrete Action Space

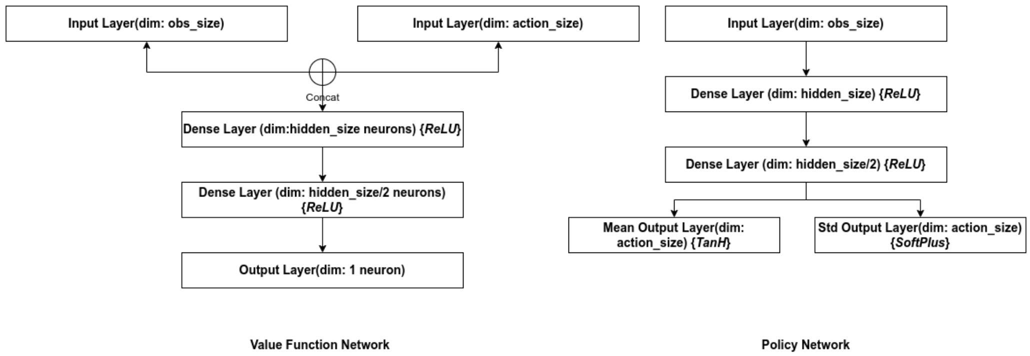

Figure 2: Networks for Continuous Action Space

1.2 Choosing Action

The policy network takes the current state as input and outputs a probability distribution over actions. The agent chooses an action based on this distribution.

1.3 Value Function update

The value function network is updated using the Q-learning algorithm, which involves computing the TD error and updating the value estimates for the current state-action pair.

1.4 Policy update

During each iteration of the algorithm, the actor selects an action based on the current state and the policy network. The Q-value of the chosen action is then computed using the critic network, which estimates the expected cumulative reward for taking the chosen action in the current state.

The policy network is then updated using the Q-value function to increase the probability of selecting the chosen action in the future. This is done by computing the gradient of the log probability of the chosen action with respect to the actor parameters and multiplying it by the Q-value. The resulting gradient is used to update the actor parameters using gradient ascent, which increases the probability of selecting the chosen action in the future.

By updating the policy network based on the Q-value function, the actor learns to select actions that are likely to lead to higher cumulative rewards in the future. This allows the algorithm to balance exploration and exploitation, exploring new actions to discover potentially high-reward options while also exploiting actions that have already been found to be effective.

Actor Loss = -1 * (log probability of action)* (TD-Error)

The negative value in the actor loss function is included because the goal of the optimization problem is to maximize the expected cumulative reward over time, which is equivalent to minimizing the negative of the expected cumulative reward. Therefore, the loss function is expressed as a negative quantity that needs to be minimized.

The first term in the actor loss function, which is the negative log-likelihood of the chosen action, is included to encourage the policy network to increase the probability of selecting actions that have higher expected cumulative rewards. By minimizing the negative log-likelihood, the policy network is encouraged to increase the probability of selecting actions that are more likely to lead to higher cumulative rewards.

The second term in the actor loss function, which is the TD-Error of the chosen action, is included to provide information on the value of the chosen action and encourage the policy network to select actions that are likely to lead to higher cumulative rewards. By minimizing the negative of the TD-Error, the policy network is encouraged to select actions that have higher expected cumulative rewards. We have observed that using TD-Error instead of Q-Value here leads to faster convergence. Also, the TD-Error is detached so that it doesn't influence the learning of the value function network.

1.5 Other Implementation details

We have chosen to update both the actor and critic networks every timestep of the environment (similar to SGD).

2 Difference between Actor-Critic and Value-based approximation algorithms

The main difference between value-based and actor-critic algorithms lies in their approach to learning and updating the policy.

Value-based algorithms, such as Q-learning and SARSA, learn the optimal value function for each state-action pair in the environment. The value function estimates the expected cumulative reward that the agent can achieve by following a certain policy. Once the value function is learned, the policy can be derived by selecting the action with the highest expected cumulative reward in each state. Value-based algorithms update the value function using a temporal-difference (TD) learning approach, where the value of the current state-action pair is updated based on the value of the next state-action pair and the reward obtained.

On the other hand, actor-critic algorithms learn both a policy (actor) and an approximate estimate of the value function (critic). The critic network in actor-critic algorithms approximates the value function, which provides an estimate of the expected cumulative reward that the agent can achieve by following the current policy. The policy network (actor) selects actions based on the output of the critic network, which is used to estimate the expected cumulative reward for each action in the current state. The critic network is updated using a TD learning approach, where the estimated value of the current state-action pair is updated based on the estimated value of the next state-action pair and the reward obtained. The policy network (actor) is updated using the gradient of the expected cumulative reward with respect to the policy parameters.

The key difference between the two approaches is that value-based algorithms only learn the value function, while actor-critic algorithms learn both the value function and the policy. The policy in actor-critic algorithms is updated based on the value function, allowing the agent to balance exploration and exploitation and learn a more optimal policy in complex environments. By learning both the policy and the value function, actor-critic algorithms can achieve better performance and faster convergence than value-based algorithms in many reinforcement learning tasks.

3 CartPole-v1

In the cart-pole problem version described by Barto, Sutton, and Anderson [1] a pole is connected to a cart through a joint that cannot be moved. The cart can move on a track without any friction. The pole is positioned upright on the cart and the objective is to keep the pole balanced by applying forces to the cart in either the left or right direction. This environment has been visualized in Figure 3.

3.1 Action Space

The action is an array of shape (1,) that can take on values 0, 1 to indicate the the direction in which the cart is pushed with a fixed force.

- 0: The cart is pushed to the left
- 1: The cart is pushed to the right

The velocity that is either decreased or increased by the force applied is not constant and depends on the angle at which the pole is pointing. The center of gravity of the pole affects the amount of energy required to move the cart beneath it.

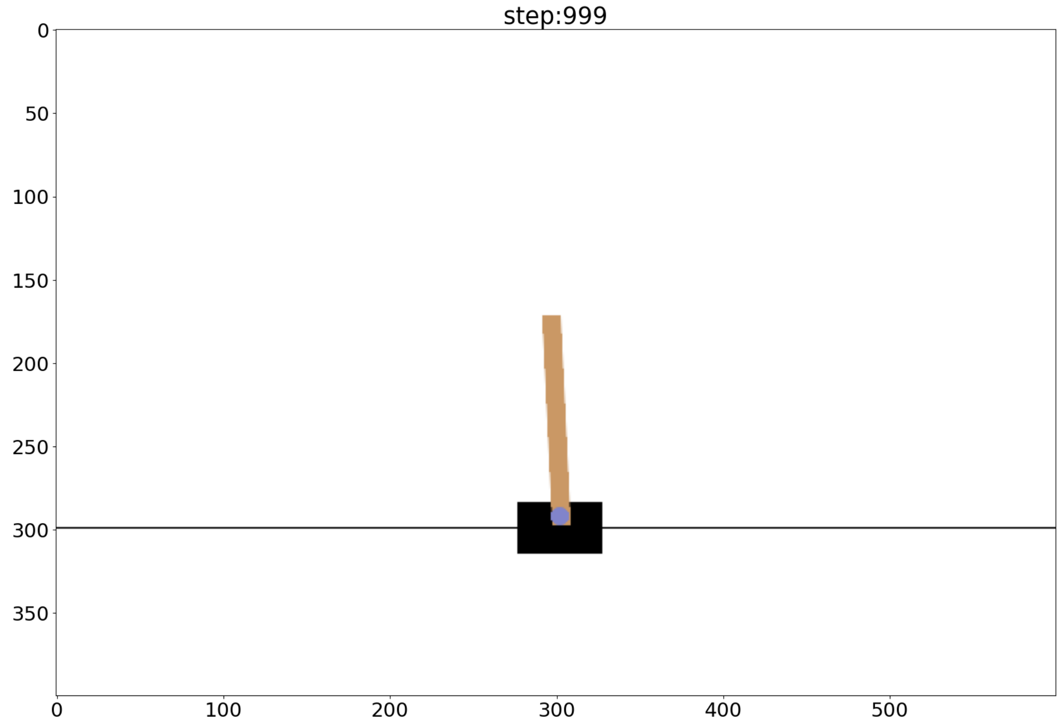

Figure 3: A frame from CartPole-v1

3.2 Observation Space

The observation is an ndarray of shape (4,) where the values represent the positions and velocities described next.

Num	Observation	Min	Max
0	Cart Position	-4.8	4.8
1	Cart Velocity	-Inf	Inf
2	Pole Angle	$\sim$ -0.418 rad (-24°)	$\sim$ 0.418 rad (24°)
3	Pole Angular Velocity	-Inf	Inf

Although the ranges above indicate the possible values for each element in the observation space, they do not reflect the allowed values of the state space in an ongoing episode. Specifically:

- The cart x-position (index 0) can be take values between `(-4.8, 4.8)`, but the episode terminates if the cart leaves the `(-2.4, 2.4)` range.
- The pole angle can be observed between `(-.418, .418)` radians or `(±24°)`, but the episode terminates if the pole angle is not in the range `(-.2095, .2095)` or `(±12°)`

3.3 Rewards

The objective of the task is to keep the pole upright for as long as possible. To encourage this behavior, a reward of +1 is given for every step taken, including the final step when the episode terminates. In version 1 of the task, the threshold for achieving a successful outcome is set at 475.

3.4 Start State

All observations are assigned a uniformly random value in `(-0.05, 0.05)`

3.5 Episode End

The episode terminates under these conditions:

1. Termination: Pole Angle is greater than `±12°`
2. Termination: Cart Position is greater than `±2.4` (center of the cart reaches the edge of the display)
3. Truncation: Episode length is greater than `500`

4 LunarLander-v2

This environment represents a classic problem of optimizing rocket trajectory. Based on Pontryagin's maximum principle, the optimal approach is to either fire the engine at full throttle or turn it off completely. As a result, this environment has discrete actions: the engine is either on or off.

Two versions of the environment are available: discrete and continuous. In our work we have used the discrete version. The landing pad is always located at coordinates `(0,0)`, which are represented by the first two numbers in the state vector. It is possible to land outside of the landing pad. Since fuel is unlimited, an agent can learn to fly and land successfully on its first attempt.

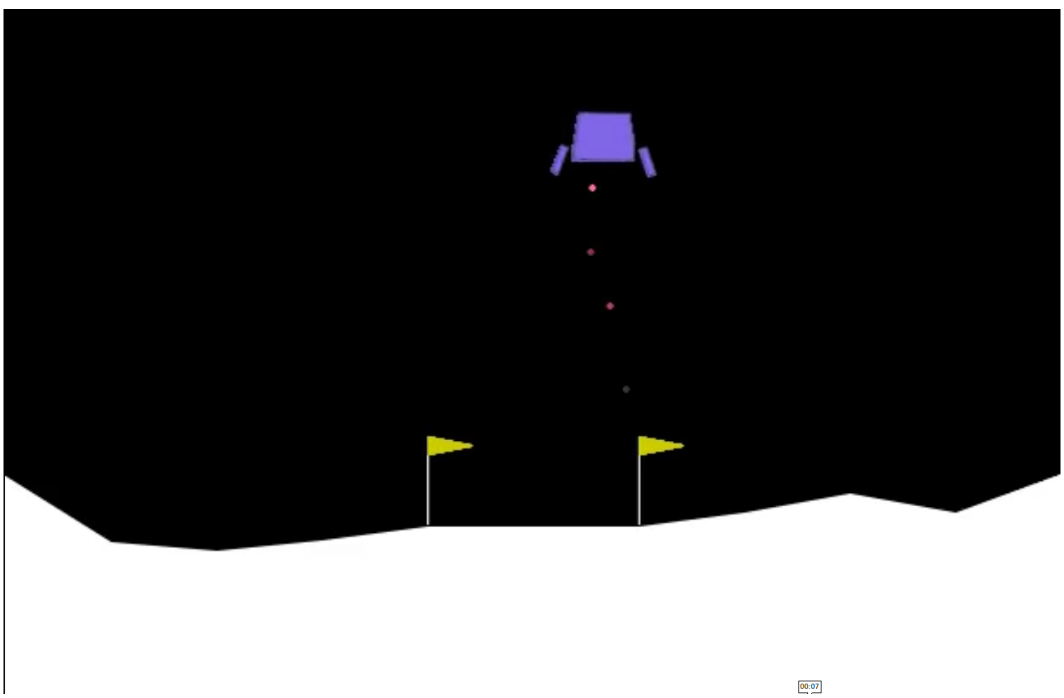

Figure 4: A frame from LunarLander-v2

4.1 Observation Space

The state of the environment is represented by an 8-dimensional vector that includes the `x` and `y` coordinates of the lander, its linear velocities in `x` and `y`, its angle and angular velocity, and two boolean values indicating whether each leg is in contact with the ground.

4.2 Action Space

Four discrete actions are available in this environment: remain idle, activate the left orientation engine, activate the main engine, or activate the right orientation engine.

4.3 Reward

A reward is given after each step in the environment. The total reward for an episode is calculated by summing the rewards for all steps within that episode. The reward for each step is determined by the following factors:

- The reward increases/decreases as the lander gets closer/further from the landing pad.
- The reward increases/decreases as the lander moves slower/faster.
- The reward decreases as the lander tilts more (angle not horizontal).
- The reward increases by 10 points for each leg in contact with the ground.
- The reward decreases by 0.03 points for each frame a side engine is firing.
- The reward decreases by 0.3 points for each frame the main engine is firing.

An additional reward of -100 or +100 points is given for crashing or landing safely, respectively. An episode is considered solved if it scores at least 200 points.

4.4 Starting State

At the beginning of each episode, the lander is positioned at the top center of the viewport and a random initial force is applied to its center of mass.

4.5 Episode End

An episode terminates if any of the following conditions are met:

1. The lander crashes (its body comes into contact with the moon).
2. The lander moves outside of the viewport (its `x` coordinate is greater than 1).
3. The lander is not awake. According to the Box2D documentation, a body that is not awake does not move or collide with any other body.

5 InvertedPendulum-v4

This environment is the cart pole environment based on the work done by Barto, Sutton, and Anderson [1]. This has a similar problem description to the classic environment but is powered by the Mujoco physics simulator - allowing for more complex experiments (such as varying the effects of gravity). This environment involves a cart that can move linearly, with a pole fixed on it at one end and another end free. The cart can be pushed left or right, and the goal is to balance the pole on the top by applying forces on the cart.

5.1 Observation Space

The state of the environment is represented by a 4-dimensional vector that includes the following.

Num	Observation	Min	Max
0	Cart Position	-Inf	Inf
1	Pole Vertical Angle	-Inf	Inf
2	Cart Linear Velocity	-Inf	Inf
3	Pole Angular Velocity	-Inf	Inf

5.2 Action Space

The agent takes a 1-dimensional vector for actions.

The action space is a continuous (action) in the range [-3, 3], where action represents the numerical force applied to the cart (with magnitude representing the amount of force and sign representing the direction)

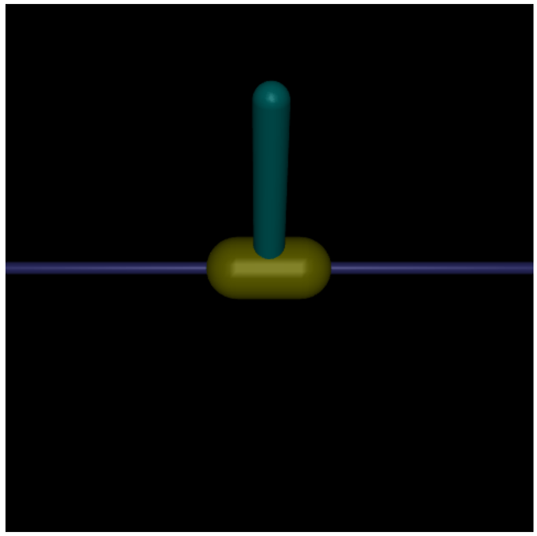

Figure 5: A frame from InvertedPendulum-v4

5.3 Reward

The goal is to make the inverted pendulum stand upright (within a certain angle limit) as long as possible - as such a reward of +1 is awarded for each timestep that the pole is upright.

5.4 Starting State

All observations start in the state (0.0, 0.0, 0.0, 0.0) with a uniform noise in the range of [-0.01, 0.01] added to the values for stochasticity.

5.5 Episode End

The episode terminates when any of the following happens:

1. The episode duration reaches 1000 timesteps.

2. Any of the state space values is no longer finite.

3. The absolute value of the vertical angle between the pole and the cart is greater than 0.2 radians.

6 Training and Evaluation

6.1 Results

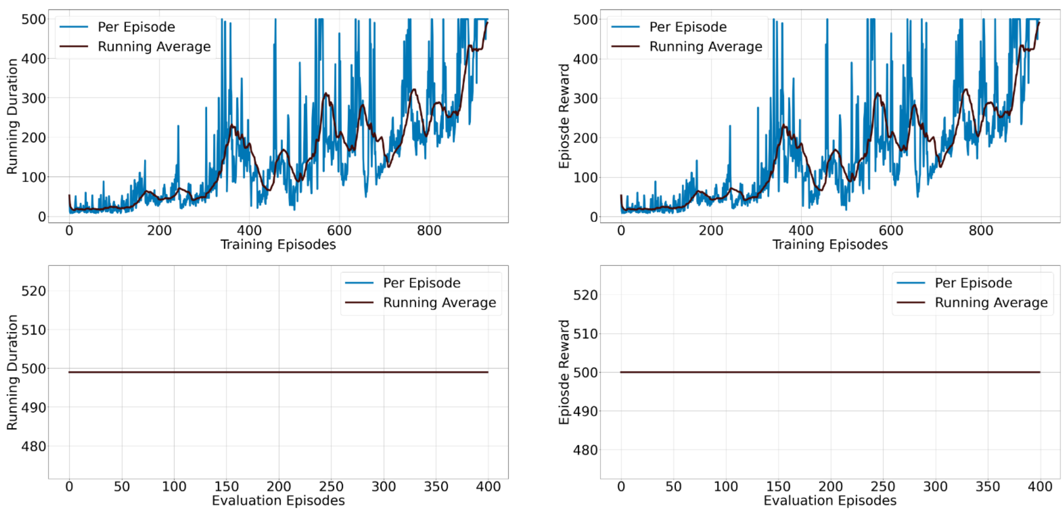

Figure 6: Cart Pole Training and Evaluation

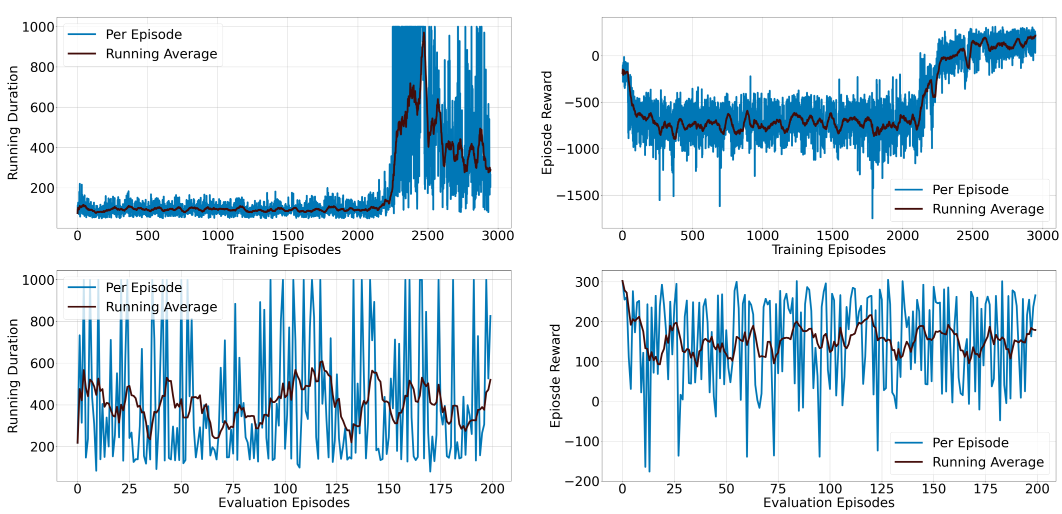

Figure 7: Lunar Lander Training and Evaluation

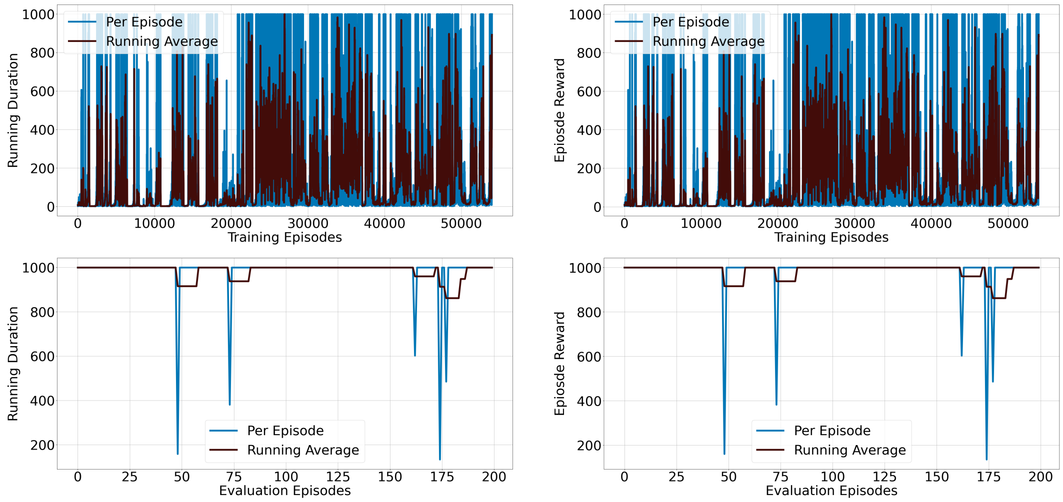

Figure 8: Inverted Pendulum Training and Evaluation

6.2 Conclusion

7 Bonus - A2C

7.1 Results

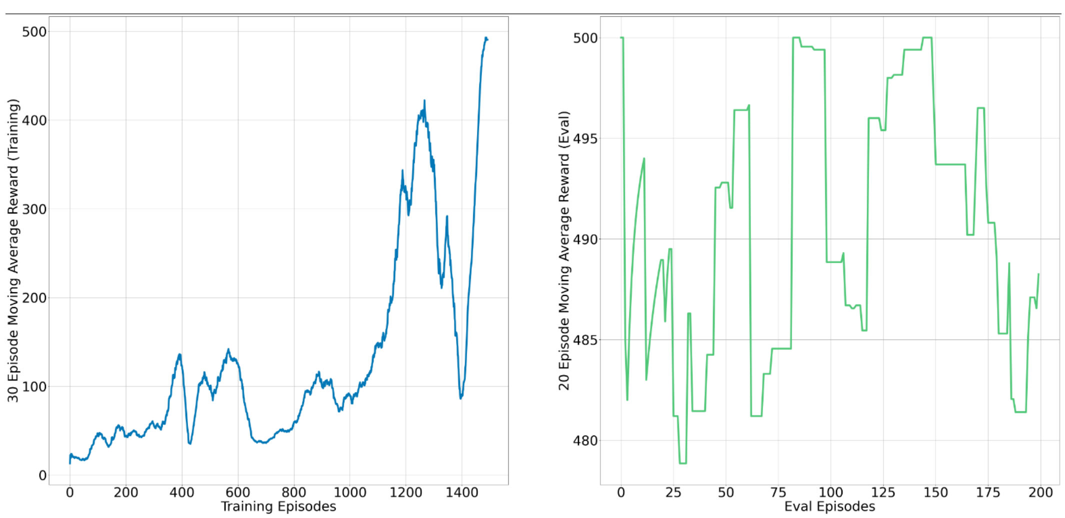

Figure 9: Cart Pole Training and Evaluation

a2c_lunar.png

Figure 10: Lunar Lander Training and Evaluation

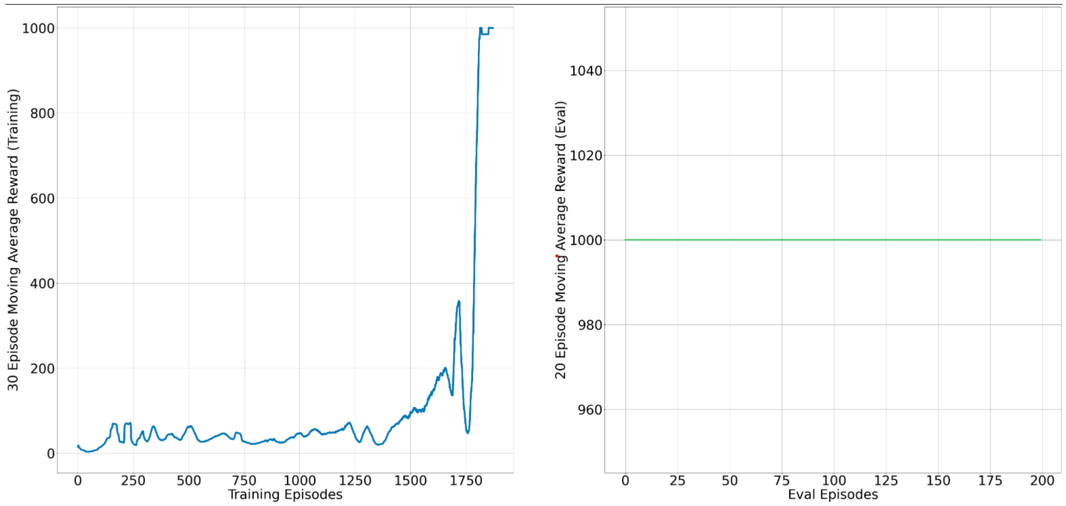

Figure 11: Inverted Pendulum Training and Evaluation

7.2 Comparision to QAC

References

[1] Andrew G Barto, Richard S Sutton, and Charles W Anderson. "Neuronlike adaptive elements that can solve difficult learning control problems". In: *IEEE transactions on systems, man, and cybernetics* 5 (1983), pp. 834–846.

[2] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: `arXiv:1606.01540`.