Actor Critic

Nandini Chinta
SUNY Buffalo, NY
nandinic@buffalo.edu

Jaishyam Ramalingam Balasubramanian Rao SUNY Buffalo, NY
jaishyam@buffalo.edu

Abstract

This project deals with applying Actor Critic algorithm on two different open ai gym environments, Lunar Lander and Cartpole and one custom grid environment.

1. Actor Critic

Value Based:

- Here we compute the value of the state and an action which is the expected reward from that state for that action.
- The states may or may not be discrete. But can also be continuous.
- Here, the policy is deterministic. For tabular methods, we use epsilon greedy policy for training, and we take best action from the table for evaluation.

Policy Based:

- Policy based algorithms are to choose the appropriate action. Here, the agent tries to converge at optimal policy instead of calculating state values.
- Policy is stochastic.
- These algorithms optimize the policy of the environment.
- We will have probability estimation of actions for each state.

Actor Critic:

Actor Critic is mixed of both Value based and Policy gradient. Actor is policy based and Critic is Value based. Actor learns the policy and Critic evaluates the policy. So, basically, we have two networks one for Actor to learn the policy and the other for Critic to evaluate the action. Replay buffer is not mandatory for Actor Critic so, it is sample inefficient. There are various versions of Actor Critic i.e. Q Actor Critic, TD Actor Critic, Advantage Actor Critic, Asynchronous Actor Critic. In current project, we have implemented Advantage Actor Critic with entropy regularization to overcome overfitting and avoid the agent to get stuck in local minima.

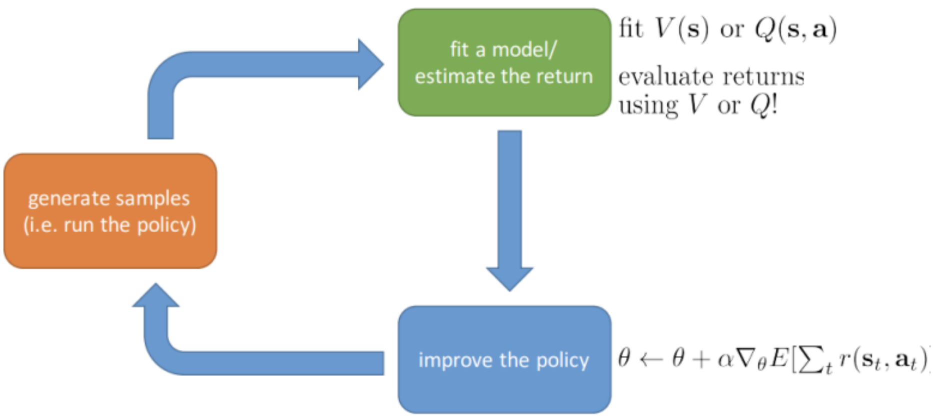
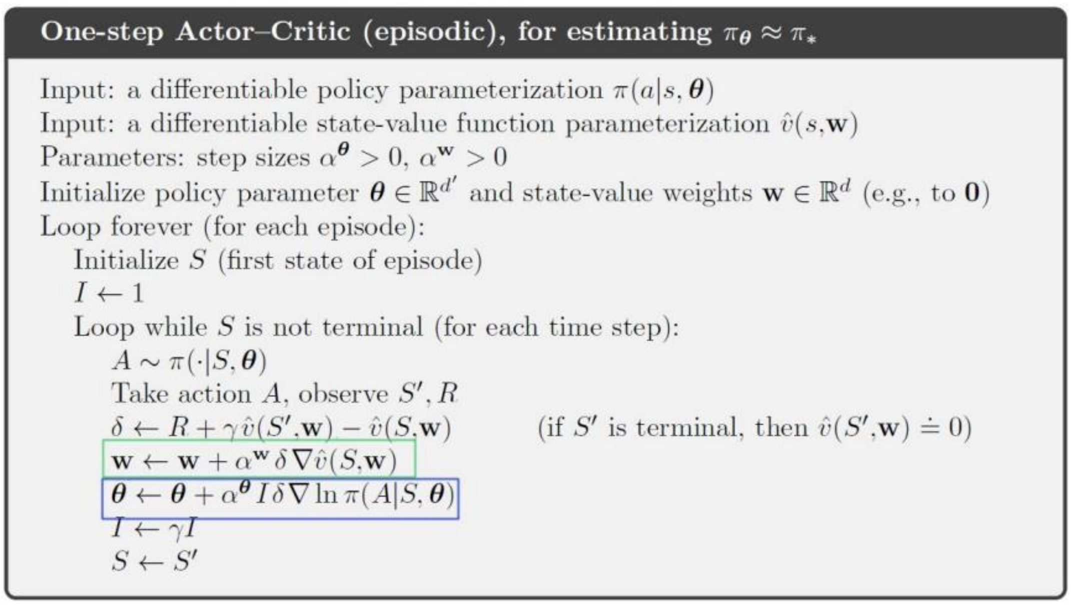

fit a model/ estimate the return
fit $V(\mathbf{s})$ or $Q(\mathbf{s},\mathbf{a})$
evaluate returns using V or Q!
generate samples (i.e. run the policy)
improve the policy
$\theta \leftarrow \theta + \alpha \nabla_\theta E[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)]$

Fig 1.1: Actor Critic

Below is the algorithm for applying Actor Critic:



One-step Actor–Critic (episodic), for estimating $\pi_{\boldsymbol\theta} \approx \pi_*$

```
Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol\theta)$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol\theta} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol\theta \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
    Initialize $S$ (first state of episode)
    $I \leftarrow 1$
    Loop while $S$ is not terminal (for each time step):
        $A \sim \pi(\cdot|S,\boldsymbol\theta)$
        Take action $A$, observe $S', R$
        $\delta \leftarrow R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$        (if $S'$ is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
        $\boxed{\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})}$
        $\boxed{\boldsymbol\theta \leftarrow \boldsymbol\theta + \alpha^{\boldsymbol\theta} I \delta \nabla \ln \pi(A|S,\boldsymbol\theta)}$
        $I \leftarrow \gamma I$
        $S \leftarrow S'$
```

Fig 1.2: Actor Critic Algorithm

In this project, we have implemented Advantage Actor Critic algorithm. Advantage is the difference between returns and the value of that state. Advantage function reduces the variance of policy gradient. Also, to avoid overfitting, entropy was introduced into the algorithm.

$$\mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A_w(s,a)]$$

Fig 1.3 : Advantage Objective Expectation

$$A(s,a) = Q_w(s,a) - V_v(s)$$

Fig 1.4: Advantage Function

2. Environments

a) Cartpole-v1

Cart-pole system with continuous actions. We have an unactuated joint attached to a cart which can go linearly on a 1-D frictionless track. Our goal is to keep the pole upright with a degree of less than 12 from the vertical and the cart should not move beyond 2.4.

Source:
This environment of the cart-pole problem was first described by Barto, Anderson, and Sutton.

Observation:

The state of the environment consists of the cart's x-position, cart's velocity, pole's angle in radians, and the angular velocity in rads/sec.

No.	Observation	Minimum	Maximum
1	Cart's Position	-4.8	4.8
2	Cart's Velocity	-Infinity	Infinity
3	Pole Angle	-0.418 rad(-24 deg)	0.418 rad(24 deg)
4	Pole Angular Velocity	-Infinity	Infinity

Actions:
{0,1}. 0 action takes the cart to the left and 1 takes it to the right.

Reward:

The reward is -1 for every step taken, and 0 if the cart-pole system is balanced. We want to motivate the agent to quickly reach the balancing.

Starting State:

At the start, the cart is upright, and the pole lies at 20 degrees and has no velocity.

Episode Termination:

Ends in Failure:

pole angle > 12 degrees cart position > 2.4 (out of bounds) episdoe length > 200
Solved:
average return > 195 over 100 consecutive trials.

b) LunarLander-v2

This environment explores the simulation of landing a space craft on the lunar surface. There are three engines on the bottom, both the sides of the spacecraft. The environment has discrete actions: engine on or off. There are two environment versions: on or off.
The landing pad is at the origin (0,0). The state vector is the co-ordinate position of the spacecraft in that frame. Fuel is infinite but each firing is penalized with a goal to help learn quick landings.

Observation space
There are in total 8 states:

- `x-y`: cartesian coordinates of the lander
- `Vx-Vy`: Linear Velocities of the lander
- `theta`: Orientation angle of the lander
- 'Atheta': Angular velocities
- 'Cl,Cr': Booleans that say if lander's left and right legs are in contact with the ground.

Action Space

There are four discrete actions available: do nothing, fire left orientation engine, fire main engine, fire right orientation engine

Rewards
The reward function is defined as:

- Top of page to landing: 100-140 points
- If moving away from landing pad, it loses reward
- In case of a crash a reward of -100 is given.
- Each leg in contact with the ground is given +10 reward.
- Firing the main engine is -0.3 points for each frame and the side engines a penalty of -0.03 points.
- Correctly landing on the landing pad would be given a reward of +200 points.

Starting State
The lander starts at the vertical top in the centre.

`Episode Termination`
The episode finishes if:
1) When the lander crashes into the surface.
2) the lander's x co-ordinate is > 1. (out of bounds)
3) the lander is not awake. (i.e. it doesn't move or do anything)

c) GridWorld

The environment created in this project is based on the movie series “Matrix”. The environment is a 4x4 grid world. The goal of the agent is to exit the “Matrix”. In the journey of our agent, it meets with some characters. Some characters in our environment will reward the agent, while other characters will penalize our agent.

The grid environment is of size 4x4; thus, it has 16 states. The agent can take four actions i.e., up, down, left, right. The agent will begin at state 0x0. At the state 0x2 the agent will meet the character Morpheus. At the state 3x0 the agent will meet the character Oracle. At the state 2x3 the agent will meet the evil character, Agent Smith. At the state 3x3 the agent will solve the environment by escaping the matrix.

The goal of the agent is to escape the Matrix. So, the reward for solving the environment is +10. If the agent meets Morpheus, the reward is +5. If the agent meets Oracle, the reward is +2. If the agent meets Agent Smith, the reward is -10. The reward other states are -0.25. The transition probability if this environment is 1.

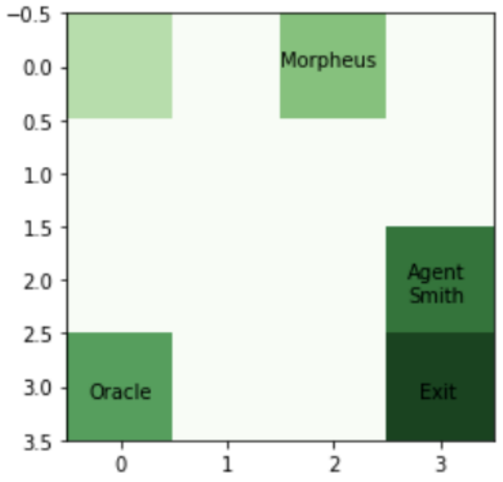


Fig 2.1: Grid World

Table 1: Description of Deterministic Environment

Parameter	Value
States	16 state grid world
Actions	Up, Down, Left, Right
Rewards	-0.25,0, +2, +5, +10, -10
Transition Probability	1

3. Actor Critic

4. Evaluation

After training the model we have achieved optimal policy in 8 timesteps. After iterating the test model for 5 epochs, we have got following results.

a) GridWorld

Training:

We have trained the model for 2000 episodes and the agent got converged and able to reach the goal.

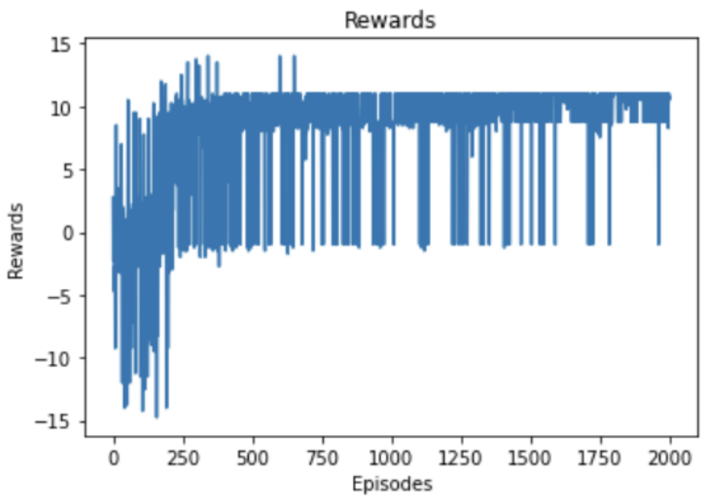


Fig 4.1: Grid World Training

Evaluation:
We can see that the agent is reaching the goal for all 10 given episodes with an average reward of 10.

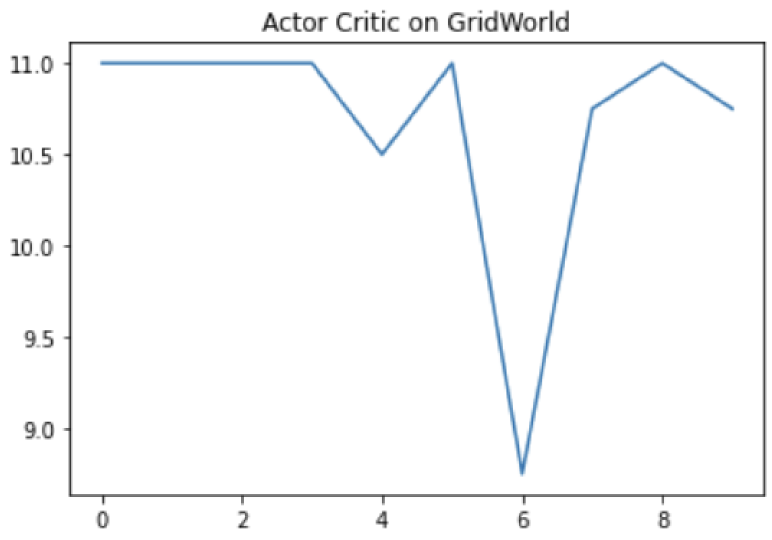


Fig 4.2: Rewards for each episode - evaluation

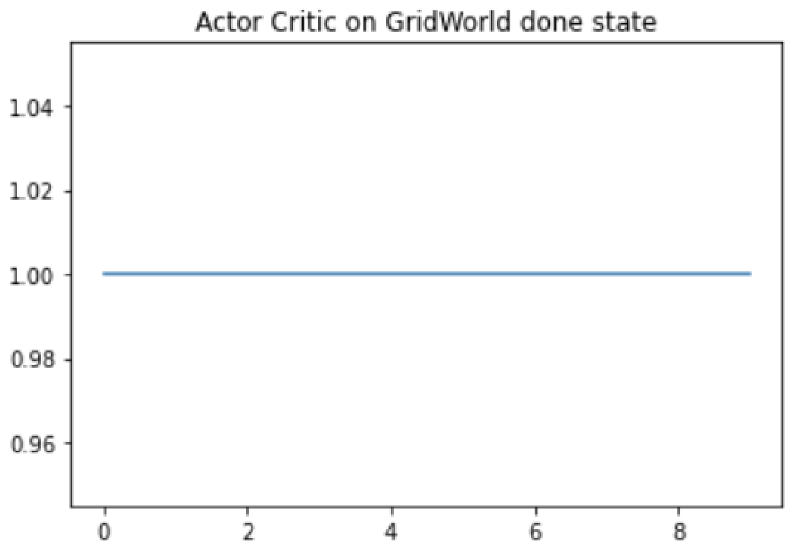
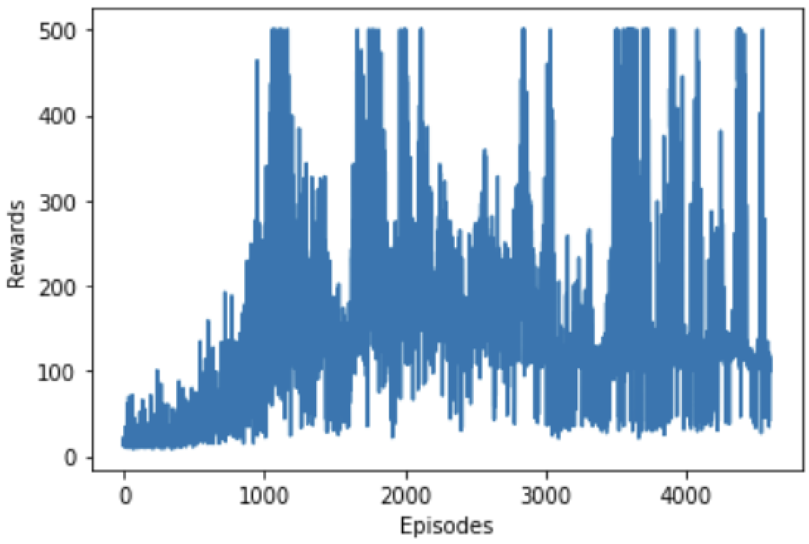


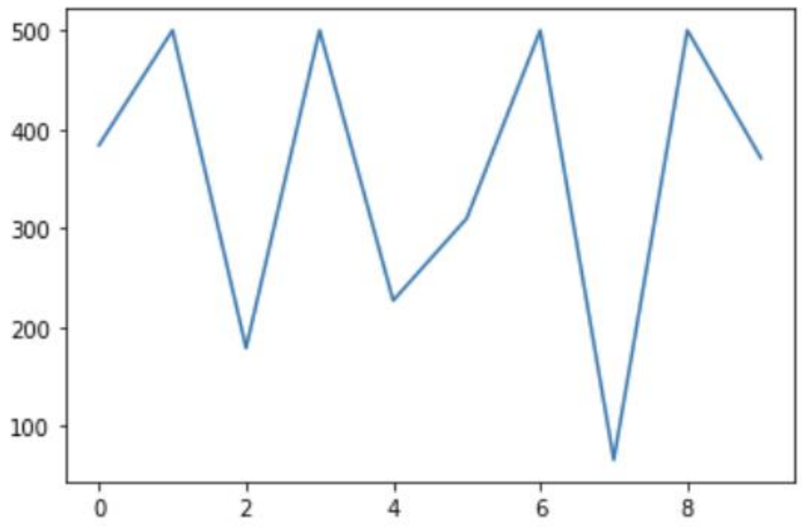
Fig 4.3: Goals Reached for each episode - evaluation

b) Cartpole-v1

Training(4600 episodes):



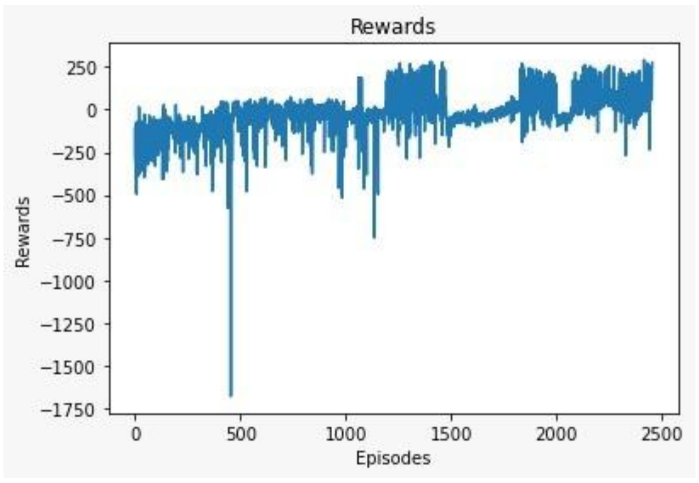
Evaluation(10 episodes):



c) LunarLander-v2

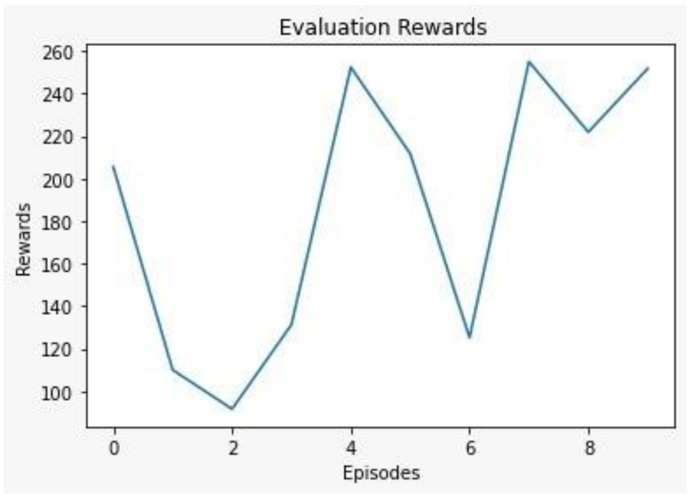
Training(2500

episodes):



Evaluation(10

episoes):



5. Conclusion

To conclude, we have implemented Advantage Actor Critic algorithm on a custom grid world, Cartpole and LunarLander environments and the agents got converged as above mentioned. Since A2C is a sample inefficient it took longer period to converge to optimal policy when compared to tabular methods.

We certify that the code and data in this assignment were generated independently, using only the tools and resources defined in the course and that I did not receive any external help, coaching, or contributions during the production of this work.

References

[1] https://gym.openai.com/docs/
[2] https://en.wikipedia.org/wiki/Reinforcement_learning
[3] Lecture Slides
[4] https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f

Contribution

Team Member	**Contribution**
Jaishyam Ramalingam Balasubramanian Rao	**50%**
Nandini Chinta	**50%**