Author:

Naga Kiran Reddy

I have implemented **TD actor-critic method**, wherein I defined 3 separate networks actor,critic, and policy.

Actor is used to take an action based on the decision given by critic network. Policy network is used to predict probabilities to act by actor network.

Main networks used:

```
input = Input(shape=(16,))
        TD_error = Input(shape=[1])
        hidden_d1 = Dense(1024, activation='relu')(input)
        hidden_d2 = Dense(512, activation='relu')(hidden_d1)
        values = Dense(1, activation='linear')(hidden_d2) #for criti
c model
        probs = Dense(self.n_actions, activation='softmax')(hidden_d
2) #output of a actor model, policy model
```

Actor network: output is probabilities

```
        # Defining an Actor model
        actor_model = Model([input, TD_error], [probs])
        actor_optimizer = Adam(lr=self.alpha)
        actor_model.compile(optimizer=actor_optimizer, loss=actor_lo
ss)
```

Critic network: Outputs are state values

```
        # Defining a critic model
        critic_model = Model([input], [values])
        critic_optimizer = Adam(lr=self.beta)
        critic_model.compile(optimizer= critic_optimizer, loss='mse'
)
```

Policy network: Outputs are probabilities

```
        # Defining an additional policy network
         policy_model = Model([input], [probs])
```

TD error calculation:

```
         # get the target values
         next_critic_value = self.critic_model.predict(next_state)
         critic_value = self.critic_model.predict(state)

         # Getting TD error values to feed into the actor model
         target = reward + self.gamma*next_critic_value*(1-int(done))
         TD_error =  target - critic_value
```

Actor and critic networks fitting:

```
         actions = np.zeros((1, self.n_actions))
         actions[[0], action] = 1

         #fitting actor and critic models
         self.actor_model.fit([state, TD_error], actions, verbose=0)
         self.critic_model.fit(state, target, verbose=0)
```

1. In value-based algorithms either state value V(s) or Q(s) is learnt whereas in actor-critic policy gradients and value functions computed are used to find optimal policy, actor-critic algorithms follow an approximate policy gradient. In actor-critic two set of networks are used: a critic that updates the action-value function parameters and an actor that update policy parameters in a direction suggested by critic. In actor-critic the actions are chosen based on direct probabilities of the actions, but in value-based algorithms e-greedy approach is used.

2. Environments

1. Grid world:

Actions: {down, up, right, left}

States: {s1, s2, s3, s4, s5....s16} -> 4x4 grid

Rewards: {-2, -1, 0, 1, 2, 150}

Main Objective: To run a random agent using actor critic algorithm in deterministic environment for 10 timesteps or episodes to find the optimal path.

optimal path

Starting position: [0,0]

Target position: [3,3] #If agent reaches target position: Reward = 150

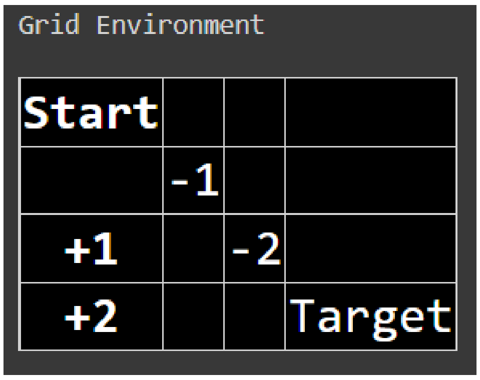
Danger1 position: [1,1] #If agent reaches danger1 position: Reward = -1

Danger2 position: [2,2] #If agent reaches danger2 position: Reward = -2

Gold1 position: [2,0] #If agent reaches gold1 position: Reward = +1

Gold2 position: [3,0] #If agent reaches gold2 position: Reward = +2

A reward of zero in all other cases. #Reward = 0


Grid Environment

Start			
	-1		
+1		-2	
+2			Target

2. Lunar Lander-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)

3. Cartpole-v1:

Cart-pole system with continuous actions. I have an unactuated joint attached to a cart which cango 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. I 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) episode length > 200

Solved:
average return > 195 over 100 consecutive trials.

4 & 5. TD Actor critic on Grid environment:

Main details:

Total_episodes = 5000

Dynamic rewards:

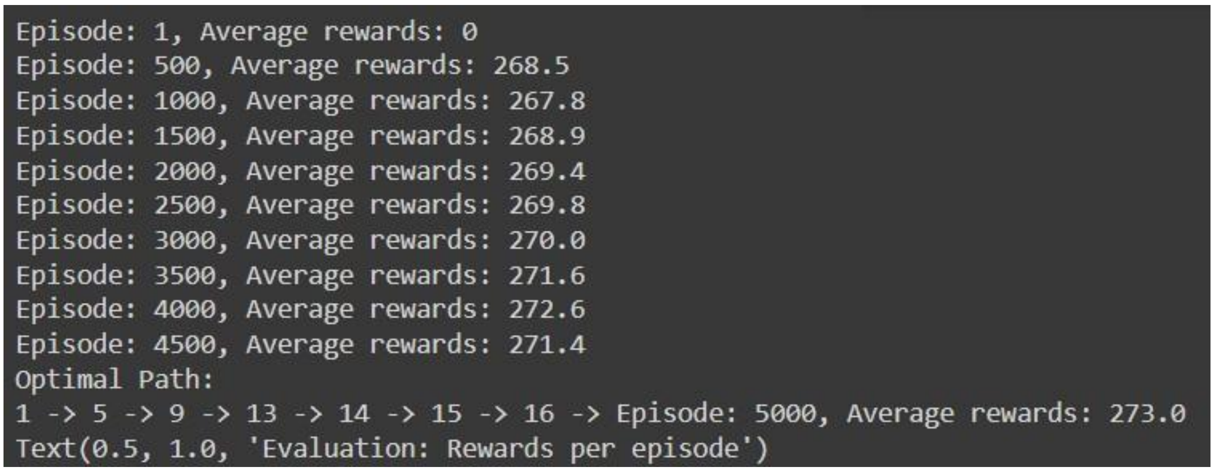
```
        #Running rewards

        if reward == 150: #if agent reaches goal
          reward += 100

        if np.linalg.norm(current - np.array(next_state_pos)) < 0.5:
 #if agent stays in same position
          reward = reward - 10
        current = np.array(next_state_pos)

        if np.linalg.norm(target - np.array(next_state_pos)) <= 1: #
if agent is in vicinity of goal position
          reward = reward + 10
```

Results:

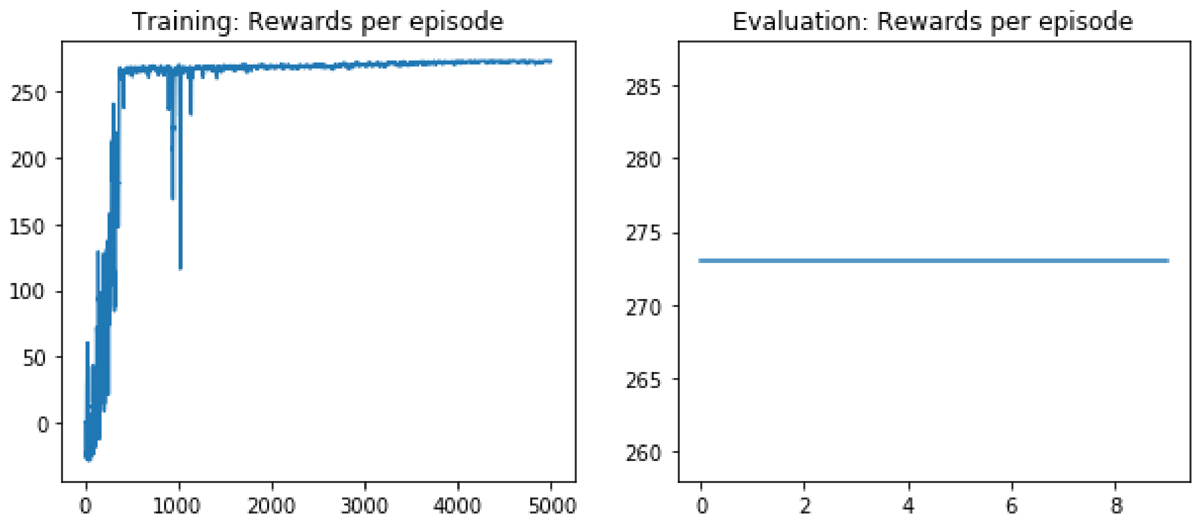


```
Episode: 1, Average rewards: 0
Episode: 500, Average rewards: 268.5
Episode: 1000, Average rewards: 267.8
Episode: 1500, Average rewards: 268.9
Episode: 2000, Average rewards: 269.4
Episode: 2500, Average rewards: 269.8
Episode: 3000, Average rewards: 270.0
Episode: 3500, Average rewards: 271.6
Episode: 4000, Average rewards: 272.6
Episode: 4500, Average rewards: 271.4
Optimal Path:
1 -> 5 -> 9 -> 13 -> 14 -> 15 -> 16 -> Episode: 5000, Average rewards: 273.0
Text(0.5, 1.0, 'Evaluation: Rewards per episode')
```

From the above results it is evident that agent is able to learn and pick optimal path

Plots:

Rewards per episodes and evaluation results



The above plots show that around 2000 episodes the agent can converge and pick optimal path while maximizing the cumulative reward.

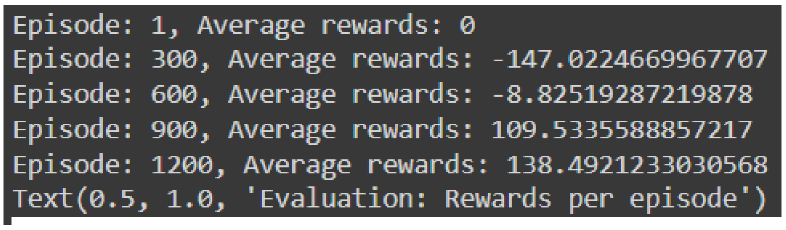
The evaluation plot also shows the agent is picking the best path giving the maximum reward over the 10 episodes.

b) Lunar Lander-v2

Total_episodes: 3000

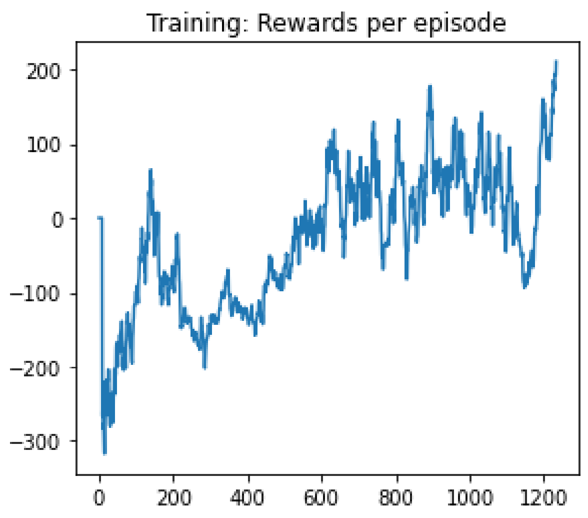
Converged around 1200 episodes.

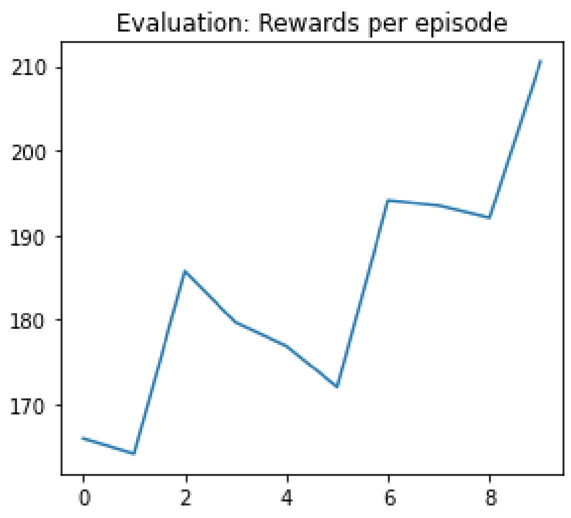
Results:



```
Episode: 1, Average rewards: 0
Episode: 300, Average rewards: -147.0224669967707
Episode: 600, Average rewards: -8.82519287219878
Episode: 900, Average rewards: 109.5335588857217
Episode: 1200, Average rewards: 138.4921233030568
Text(0.5, 1.0, 'Evaluation: Rewards per episode')
```

The agent has learnt and is able to earn the average reward of **200** in 10 consecutive episodes below plot shows this idea.





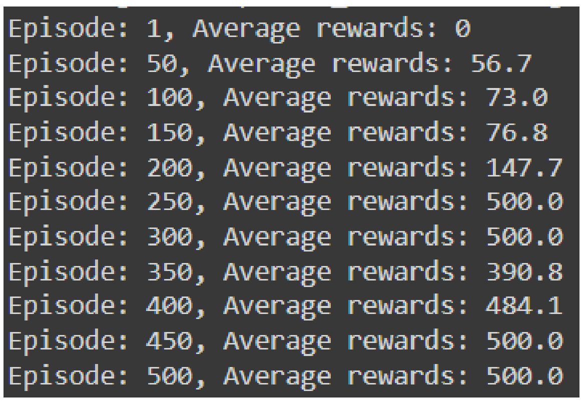
The above rewards per episode clearly shows that the agent is learning (i.e., increasing reward) and reached an average reward of 200.

The evaluation plot clearly shows that the agent has learnt and started from 170 and able to cross 200 in few episodes. (i.e., as algorithm is stopped for 200 for 10 consecutive episodes)

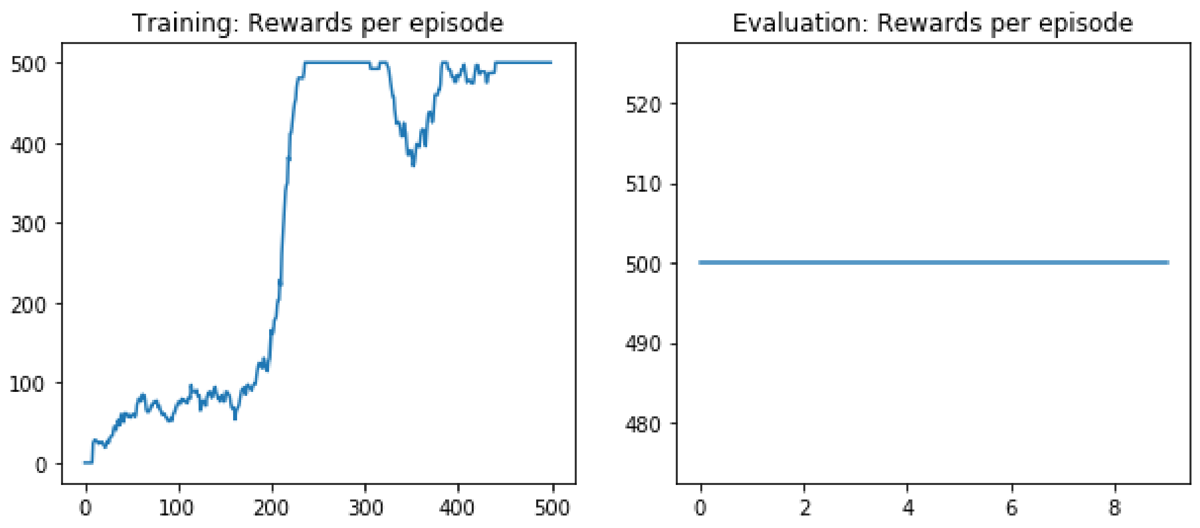
C) Cartpole-v1

Total_episodes: 500

Results:



```
Episode: 1, Average rewards: 0
Episode: 50, Average rewards: 56.7
Episode: 100, Average rewards: 73.0
Episode: 150, Average rewards: 76.8
Episode: 200, Average rewards: 147.7
Episode: 250, Average rewards: 500.0
Episode: 300, Average rewards: 500.0
Episode: 350, Average rewards: 390.8
Episode: 400, Average rewards: 484.1
Episode: 450, Average rewards: 500.0
Episode: 500, Average rewards: 500.0
```



The above plot shows the cart pole can balance for 500 episodes towards the end of the episodes, so converged.

The evaluation plot clearly shows the agent has learnt an optimal policy or optimal actions to balance the pole for 500 episodes.

Comparison:

The actor-critic algorithm performed well on grid world, converged sooner. I tried different setupsfor Lunar Lander with different hidden nodes and optimizer learning rates, small learning rates provided better results. On cartpole the actor-critic took around 450 episodes to converge on contrary to DQN in less than 100 episodes (i.e., details from assignment 2). So, actor-critic takes a lotof time exploring and converges slower compared to value-based approximation algorithms.

References:

https://github.com/philtabor/Youtube-Code-Repository