Exercise 6

Sheetal Borar - 915263
ELEC-E8125 - Reinforcement Learning

November 2, 2020

1 Task 1

I implemented actor critic TD(0) update with the following changes in the agent.py

In policy init function -

```

self.fc1 = torch.nn.Linear(state_space, self.hidden)
self.actor_layer = torch.nn.Linear(self.hidden, action_space)

# TODO: Add another linear layer for the critic
self.state_layer = torch.nn.Linear(self.hidden, 1)

self.sigma = torch.nn.Parameter(torch.tensor([20]).float())
```

In policy forward function -

```python
def forward(self, x):
    # Common part
    x = self.fc1(x)
    x = F.relu(x)

    # Actor part
    action_mean = self.actor_layer(x)
    sigma = self.sigma  # TODO: Implement (or copy from Ex5)

    # Critic part
    # TODO: Implement
    state_value = self.state_layer(x)

    # TODO: Instantiate and return a normal distribution
    # with mean mu and std of sigma
    # Implement or copy from Ex5
    action_dist = Normal(action_mean, sigma)

    # TODO: Return state value in addition to the distribution
    return action_dist, state_value
```

Agent update policy function

```
# TODO: Compute state values
state_value_approxs = self.policy.forward(states)[1].squeeze
    (-1)
next_state_value_approxs = self.policy.forward(next_states)[1].
    squeeze(-1)

# # TODO: Compute critic loss (MSE)
true_state_value  = rewards + (1 - done) *
    next_state_value_approxs.detach() * self.gamma
critic_loss = F.mse_loss(state_value_approxs, true_state_value)

# TODO: Compute advantage estimates
advantage = true_state_value - state_value_approxs

# TODO: Calculate actor loss (very similar to PG)
actor_gradient = - torch.mean(action_probs * advantage.detach()
    )

# Compute the gradients of loss w.r.t. network parameters (T1)
loss = critic_loss + actor_gradient
loss.backward()

# Update network parameters using self.optimizer and zero
    gradients (T1)
self.optimizer.step()
self.optimizer.zero_grad()
```

Agent get action function

```python
def get_action(self, observation, evaluation=False):
    x = torch.from_numpy(observation).float().to(self.
       train_device)

    # Pass state x through the policy network
    action_dist, _ = self.policy.forward(x)

    # Return mean if evaluation, else sample from the
       distribution
    if(evaluation):
        action = action_dist.mean
    else:
        action = action_dist.sample()

    # Calculate the log probability of the action
    act_log_prob = action_dist.log_prob(action)

    return action, act_log_prob
```

Figure 1 shows the learning curve for task 1.

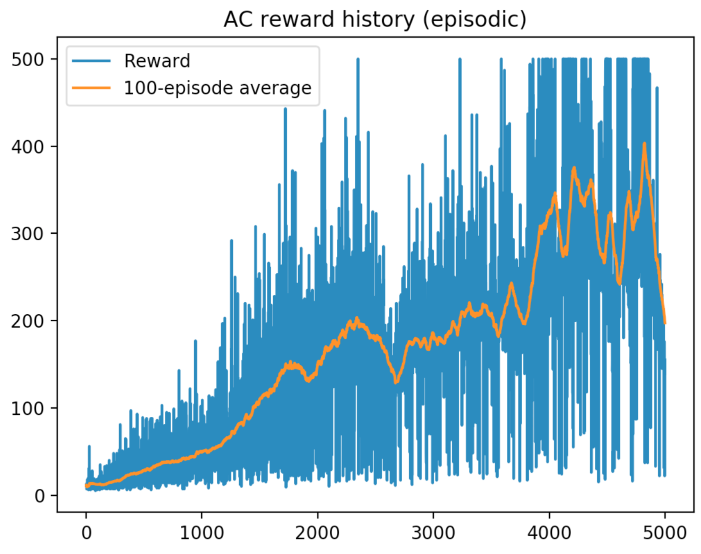

Figure 1: Learning curve for Task 1 AC episodic task

1.1 Question 1

REINFORCE with baseline uses the state function only as a baseline and not as a critic. In REINFORCE with baseline, we can use the value function as it is a good baseline, as it approximates the returns of a state. Then we have Q(s,a, θ) - V(s, w) in the gradient, which is the advantage function used in Actor critic methods. The difference is that AC uses the advantages of bootstrapping to estimate the long term rewards of a state instead of waiting till the end of the episode to get the true rewards.

1.2 Question 2

Advantage can be intuitively interpreted as the advantage one can get by choosing a particular action in a particular state. value function is the average reward one is expected to get in a state. Advantage tells us how much above/below average is it to take action a in state s.

2 Task 2

Outside the training for loop, i defined a new variable called total time steps, and did not make it 0 when the while loop breaks. This gave me the total time steps. For every 50 time steps i updated the function.

```
total_timesteps += 1

if total_timesteps % 50 == 0:
    # print('update')
    agent.update_policy(episode_number)
```

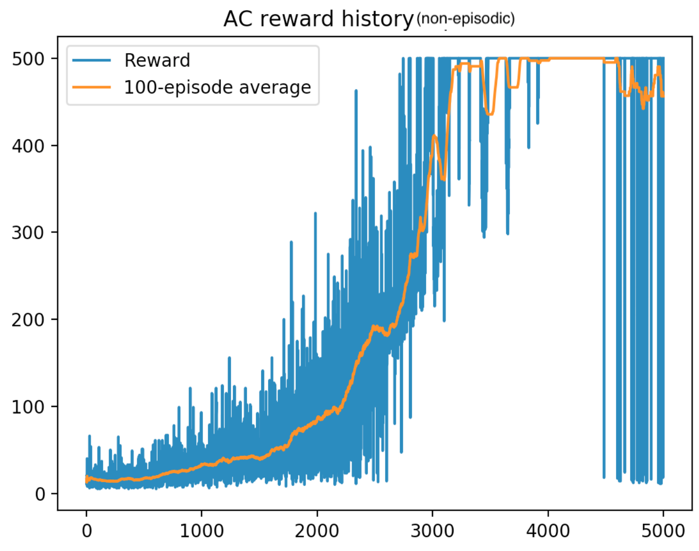

Figure 2: Learning curve for AC non episodic

3 Task 3

Figure 3 shows the learning plot for task 3 parallel AC.

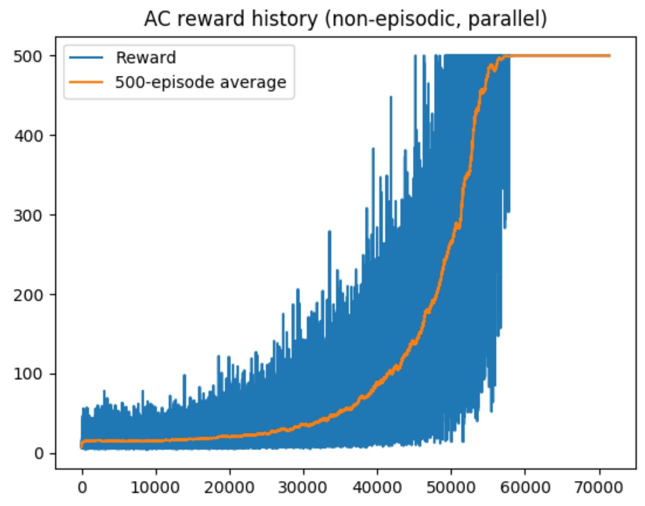

Figure 3: Learning curve for AC non episodic (parallel)

3.1 Question 3

In multiple cartpole script, there is a one-to-one mapping between agent and environment, it essentially means that we are running the many instances of the agent and updating them based on their separate environments. Whereas in parallel cartpole we are running the same instance of the agent through many environments parallelly and updating the same agent instance through these runs in many different environments.

The RL model is stochastic because of the agents initial weights(model), random initialization of the state(environment) and the policy itself is stochastic(exploration/exploitation). When we run multiple environments through the same instance of the agent, we reduce the effect of the randomness created by the environment as we are running it multiple times through many different environments. But the initialization randomness of the agent still remains and hence it is not as reliable if we use parallel cartpole to compare different agents. At the end of running multiple cartpole, we only have 1 agent trained and hence we cannot compare the performance of that agent with other agents to see how it performs on average and whats the standard deviation, hence parallel cartpole cannot replace multiple cartpole script.

3.2 Question 4

The reason we use a critic is to make sure that our model encourages actions that give higher rewards, rather than all actions in a trajectory which ended up in a high reward. But in the beginning, the critic hasn't learned the right values of all states and hence, its likely very biased towards the initialization. Hence, the critic might lead the actor in a wrong direction in the beginning. This leads to slower convergence of the actor critic methods in comparison to the REINFORCE.

3.3 Question 5.1

Actor critic methods help reduce the variance of the REINFORCE algorithm. As REINFORCE is a monte carlo algorithm and needs to run till the end of the episode it has zero bias, but very high variance. In comparison, actor critic methods can be updated in the middle of an episode with TD updates. But AC will have a higher bias than REINFORCE as we do not know if the critic is always correct as its also estimated from the model.

3.4 Question 5.2

Bias-variance trade off can be controlled using eligibility traces and doing TD(λ) update and comparing the results of using different λ. As we increase λ, the bias reduces and variance goes up.
As we saw in this exercise, we did the update after every 50 timesteps, that is also a strategy to control bias/variance trade-off. If we do an update after 1 timestep, their is high bias, where as doing an update after the entire episode there is high variance. We can choose the number of timesteps after which the update should happen as a hyper parameter, which gives us a good bias-variance trade-off.

3.5 Question 6

PG/AC methods are inherently different from action value methods like Q-learning. In Q-learning, we try to learn the Q values for each state, action pair to select the action with max Q. In PG/AC methods, we try to learn a policy that maps between state to action directly. This policy can be stochastic and can be applied to a continuous action space. The advantages of using PG/AC methods are as follows -

- Stochastic policies - Action value methods return one action which should provides the maximum return. Stochastic policies allow us to return a action distribution, where we can select an action stochasticly rather than deterministically. This allows the policy to explore more and learn in scenarios in which a deterministic policy might fail like partially observable environments.

- Continuous action spaces - In PG/AC methods, we are learning the state-action mapping and we don't need to find the action that gives the max return like the action value methods(this is a very expensive operation in continuous action spaces). Hence, its possible to use PG/AC methods in continuous action spaces, while its not feasible to use action value methods for the same.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press, 2018.

[1] [?]