Tutorial 7

● Graded

Student

Anirud N

Total Points

6 / 6 pts

Question 1

Critic Loss and update code

2 / 2 pts

✔ **– 0 pts** Correct

Question 2

Actor Loss and update code

2 / 2 pts

✔ **– 0 pts** Correct

Question 3

Reward curve plot

0 / 0 pts

✔ **– 0 pts** Correct

Question 4

Your inference

2 / 2 pts

✔ **– 0 pts** Correct

No questions assigned to the following page.

```
import numpy as np
import gym
from collections import deque
import random

# Ornstein-Ulhenbeck Process
# Taken from
#https://github.com/vitchyr/rlkit/blob/master/rlkit/exploration_strate
gies/ou_strategy.py
class OUNoise(object):
    def __init__(self, action_space, mu=0.0, theta=0.15,
max_sigma=0.3, min_sigma=0.3, decay_period=100000):
        self.mu           = mu
        self.theta        = theta
        self.sigma        = max_sigma
        self.max_sigma    = max_sigma
        self.min_sigma    = min_sigma
        self.decay_period = decay_period
        self.action_dim   = action_space.shape[0]
        self.low          = action_space.low
        self.high         = action_space.high
        self.reset()

    def reset(self):
        self.state = np.ones(self.action_dim) * self.mu

    def evolve_state(self):
        x  = self.state
        dx = self.theta * (self.mu - x) + self.sigma *
np.random.randn(self.action_dim)
        self.state = x + dx
        return self.state

    def get_action(self, action, t=0):
        ou_state = self.evolve_state()
        self.sigma = self.max_sigma - (self.max_sigma -
self.min_sigma) * min(1.0, t / self.decay_period)
        return np.clip(action + ou_state, self.low, self.high)


# https://github.com/openai/gym/blob/master/gym/core.py
class NormalizedEnv(gym.ActionWrapper):
    """ Wrap action """

    def action(self, action):
        act_k = (self.action_space.high - self.action_space.low)/ 2.
        act_b = (self.action_space.high + self.action_space.low)/ 2.
        return act_k * action + act_b
```

No questions assigned to the following page.

```
class Memory:
    def __init__(self, max_size):
        self.max_size = max_size
        self.buffer = deque(maxlen=max_size)

    def push(self, state, action, reward, next_state, done):
        experience = (state, action, np.array([reward]), next_state,
done)
        self.buffer.append(experience)

    def sample(self, batch_size):
        state_batch = []
        action_batch = []
        reward_batch = []
        next_state_batch = []
        done_batch = []

        batch = random.sample(self.buffer, batch_size)

        for experience in batch:
            state, action, reward, next_state, done = experience
            state_batch.append(state)
            action_batch.append(action)
            reward_batch.append(reward)
            next_state_batch.append(next_state)
            done_batch.append(done)

        return state_batch, action_batch, reward_batch,
next_state_batch, done_batch

    def __len__(self):
        return len(self.buffer)
```

DDPG uses four neural networks: a Q network, a deterministic policy network, a

1. List item
2. List item

target Q network, and a target policy network.

No questions assigned to the following page.

$$\text{Parameters:}$$

$$\theta^{Q} : \text{Q network}$$

$$\theta^{\mu} : \text{Deterministic policy function}$$

$$\theta^{Q'} : \text{target Q network}$$

$$\theta^{\mu'} : \text{target policy network}$$

The Q network and policy network is very much like simple Advantage Actor-Critic, but in DDPG, the Actor directly maps states to actions (the output of the network directly the output) instead of outputting the probability distribution across a discrete action space.

The target networks are time-delayed copies of their original networks that slowly track the learned networks. Using these target value networks greatly improve stability in learning.

Let's create these networks.

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.autograd
from torch.autograd import Variable

class Critic(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(Critic, self).__init__()
        self.linear1 = nn.Linear(input_size, hidden_size)
        self.linear2 = nn.Linear(hidden_size, hidden_size)
        self.linear3 = nn.Linear(hidden_size, output_size)

    def forward(self, state, action):
        """
        Params state and actions are torch tensors
        """
        x = torch.cat([state, action], 1)
        x = F.relu(self.linear1(x))
        x = F.relu(self.linear2(x))
        x = self.linear3(x)
```

No questions assigned to the following page.

```
        return x

class Actor(nn.Module):
    def __init__(self, input_size, hidden_size, output_size,
learning_rate = 3e-4):
        super(Actor, self).__init__()
        self.linear1 = nn.Linear(input_size, hidden_size)
        self.linear2 = nn.Linear(hidden_size, hidden_size)
        self.linear3 = nn.Linear(hidden_size, output_size)

    def forward(self, state):
        """
        Param state is a torch tensor
        """
        x = F.relu(self.linear1(state))
        x = F.relu(self.linear2(x))
        x = torch.tanh(self.linear3(x))

        return x
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to
`transformed_cell` argument and any exception that happen during
thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Now, let's create the DDPG agent. The agent class has two main functions: "get_action" and "update":

- **get_action()**: This function runs a forward pass through the actor network to select a determinisitic action. In the DDPG paper, the authors use Ornstein-Uhlenbeck Process to add noise to the action output (Uhlenbeck & Ornstein, 1930), thereby resulting in exploration in the environment. Class OUNoise (in cell 1) implements this.

$$\mu'(s_t) = \mu(s_t|\theta_t^\mu) + \mathcal{N}$$

- **update()**: This function is used for updating the actor and critic networks, and forms the core of the DDPG algorithm. The replay buffer is first sampled to get a batch of experiences of the form **<states, actions, rewards, next_states>**.

The value network is updated similarly as is done in Q-learning. The updated Q value is obtained by the Bellman equation. However, in DDPG, the next-state Q values are calculated with the target value network and target policy network. Then, we minimize the mean-squared loss between the updated Q value and the original Q value:

No questions assigned to the following page.

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$$

$$Loss = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$$

For the policy function, our objective is to maximize the expected return. To calculate the policy loss, we take the derivative of the objective function with respect to the policy parameter. Keep in mind that the actor (policy) function is differentiable, so we have to apply the chain rule.

But since we are updating the policy in an off-policy way with batches of experience, we take the mean of the sum of gradients calculated from the mini-batch:

$$\nabla_{\theta^\mu} J(\theta) \approx \frac{1}{N}\sum_i [\nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s=s_i}]$$

We make a copy of the target network parameters and have them slowly track those of the learned networks via “soft updates,” as illustrated below:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

$$\text{where} \quad \tau \ll 1$$

```
import torch
import torch.autograd
import torch.optim as optim
import torch.nn as nn
# from model import *
# from utils import *
```

No questions assigned to the following page.

```
class DDPGagent:
    def __init__(self, env, hidden_size=256, actor_learning_rate=1e-4,
critic_learning_rate=1e-3, gamma=0.99, tau=1e-2,
max_memory_size=50000):
        # Params
        self.num_states = env.observation_space.shape[0]
        self.num_actions = env.action_space.shape[0]
        self.gamma = gamma
        self.tau = tau

        # Networks
        self.actor = Actor(self.num_states, hidden_size,
self.num_actions)
        self.actor_target = Actor(self.num_states, hidden_size,
self.num_actions)
        self.critic = Critic(self.num_states + self.num_actions,
hidden_size, self.num_actions)
        self.critic_target = Critic(self.num_states +
self.num_actions, hidden_size, self.num_actions)

        for target_param, param in zip(self.actor_target.parameters(),
self.actor.parameters()):
            target_param.data.copy_(param.data)
            target_param.requires_grad = False

        for target_param, param in
zip(self.critic_target.parameters(), self.critic.parameters()):
            target_param.data.copy_(param.data)
            target_param.requires_grad = False

        # Training
        self.memory = Memory(max_memory_size)
        self.critic_criterion  = nn.MSELoss()
        self.actor_optimizer  = optim.Adam(self.actor.parameters(),
lr=actor_learning_rate)
        self.critic_optimizer = optim.Adam(self.critic.parameters(),
lr=critic_learning_rate)

    def get_action(self, state):
        state = Variable(torch.from_numpy(state).float().unsqueeze(0))
        action = self.actor.forward(state)
        action = action.detach().numpy()[0,0]
        return action

    def update(self, batch_size):
        states, actions, rewards, next_states, _ =
self.memory.sample(batch_size)
        states = torch.FloatTensor(states)
        actions = torch.FloatTensor(actions)
        rewards = torch.FloatTensor(rewards)
```

Questions assigned to the following page: 2 and 1

```
        next_states = torch.FloatTensor(next_states)
        current_Q_values = self.critic.forward(states, actions)
        next_actions = self.actor_target.forward(next_states)
        next_Q_values = self.critic_target.forward(next_states,
next_actions.detach())
        # Implement critic loss and update critic
        target_Q_values = rewards + self.gamma * next_Q_values
        critic_loss = self.critic_criterion(current_Q_values,
target_Q_values)
        self.critic_optimizer.zero_grad()
        critic_loss.backward()
        self.critic_optimizer.step()
        # Implement actor loss and update actor
        current_actions = self.actor.forward(states)
        current_Q_values = self.critic.forward(states,
current_actions)
        actor_loss = -current_Q_values.mean()
        self.actor_optimizer.zero_grad()
        actor_loss.backward()
        self.actor_optimizer.step()
        # update target networks
        for target_param, param in zip(self.actor_target.parameters(),
self.actor.parameters()):
            target_param.data.copy_(self.tau*param.data + (1-
self.tau)*target_param.data)

        for target_param, param in
zip(self.critic_target.parameters(), self.critic.parameters()):
            target_param.data.copy_(self.tau*param.data + (1-
self.tau)*target_param.data)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to
`transformed_cell` argument and any exception that happen during
thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Putting it all together: DDPG in action.

The main function below runs 50 episodes of DDPG on the "Pendulum-v1" environment of OpenAI gym. This is the inverted pendulum swingup problem, a classic problem in the control literature. In this version of the problem, the pendulum starts in a random position, and the goal is to swing it up so it stays upright.

Each episode is for a maximum of 500 timesteps. At each step, the agent chooses an action, updates its parameters according to the DDPG algorithm and moves to the next state, repeating this process till the end of the episode.

The DDPG algorithm is as follows:

No questions assigned to the following page.

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

```
import sys
import gym
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

env = NormalizedEnv(gym.make("Pendulum-v1"))

agent = DDPGagent(env)
noise = OUNoise(env.action_space)
batch_size = 128
rewards = []
avg_rewards = []

for episode in range(50):
    state = env.reset()
    noise.reset()
    episode_reward = 0

    for step in range(500):
```

No questions assigned to the following page.

```
        action = agent.get_action(state)
        #Add noise to action

        action = noise.get_action(action)
        new_state, reward, done, _ = env.step(action)
        agent.memory.push(state, action, reward, new_state, done)

        if len(agent.memory) > batch_size:
            agent.update(batch_size)

        state = new_state
        episode_reward += reward

        if done:
            sys.stdout.write("episode: {}, reward: {}, average
_reward: {} \n".format(episode, np.round(episode_reward, decimals=2),
np.mean(rewards[-10:])))
            break

    rewards.append(episode_reward)
    avg_rewards.append(np.mean(rewards[-10:]))

plt.plot(rewards)
plt.plot(avg_rewards)
plt.plot()
plt.xlabel('Episode')
plt.ylabel('Reward')
plt.show()
```

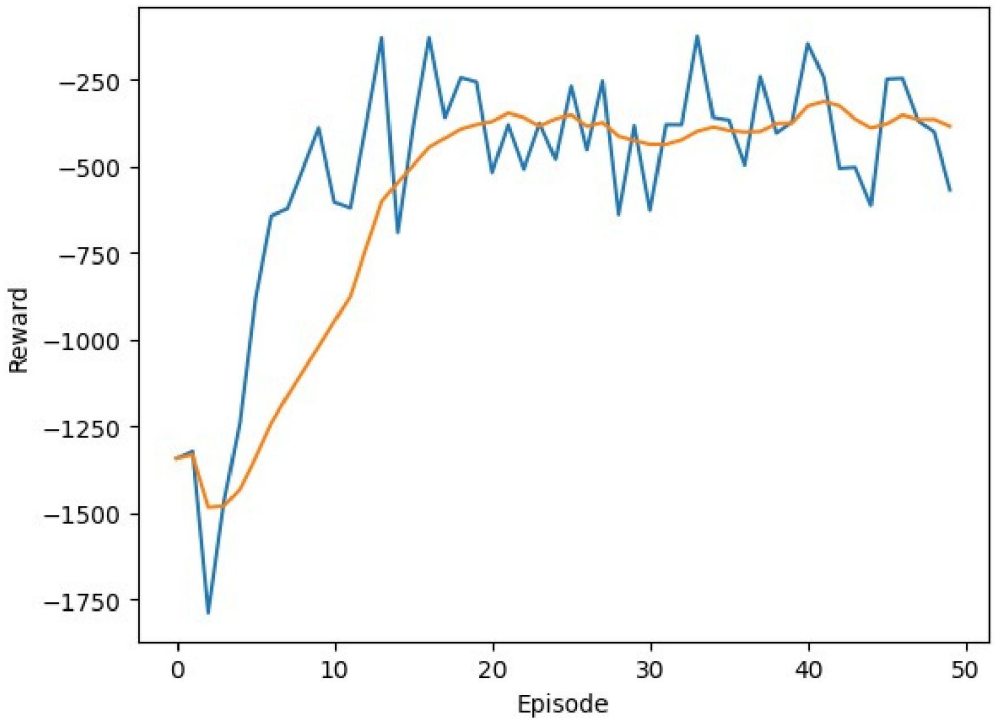
```
/usr/local/lib/python3.10/dist-packages/gym/core.py:317:
DeprecationWarning: WARN: Initializing wrapper in old step API which
returns one bool instead of two. It is recommended to set
`new_step_api=True` to use new step API. This will be the default
behaviour in future.
  deprecation(
/usr/local/lib/python3.10/dist-packages/gym/wrappers/step_api_compatib
ility.py:39: DeprecationWarning: WARN: Initializing environment in old
step API which returns one bool instead of two. It is recommended to
set `new_step_api=True` to use new step API. This will be the default
behaviour in future.
  deprecation(
/usr/local/lib/python3.10/dist-packages/gym/utils/passive_env_checker.
py:241: DeprecationWarning: `np.bool8` is a deprecated alias for
`np.bool_`.  (Deprecated NumPy 1.24)
  if not isinstance(terminated, (bool, np.bool8)):
<ipython-input-3-671bcf98c4b4>:44: UserWarning: Creating a tensor from
a list of numpy.ndarrays is extremely slow. Please consider converting
the list to a single numpy.ndarray with numpy.array() before
converting to a tensor. (Triggered internally at
../torch/csrc/utils/tensor_new.cpp:275.)
```

No questions assigned to the following page.

```
  states = torch.FloatTensor(states)
/usr/local/lib/python3.10/dist-packages/numpy/core/fromnumeric.py:3504
: RuntimeWarning: Mean of empty slice.
  return _methods._mean(a, axis=axis, dtype=dtype,
/usr/local/lib/python3.10/dist-packages/numpy/core/_methods.py:129:
RuntimeWarning: invalid value encountered in scalar divide
  ret = ret.dtype.type(ret / rcount)

episode: 0, reward: -1343.78, average _reward: nan
episode: 1, reward: -1323.43, average _reward: -1343.7802034368444
episode: 2, reward: -1788.5, average _reward: -1333.6058638176135
episode: 3, reward: -1468.83, average _reward: -1485.2372480915085
episode: 4, reward: -1240.86, average _reward: -1481.1346479632293
episode: 5, reward: -881.96, average _reward: -1433.0790391107894
episode: 6, reward: -643.83, average _reward: -1341.2263222155086
episode: 7, reward: -623.56, average _reward: -1241.5986170820204
episode: 8, reward: -507.01, average _reward: -1164.3444142100857
episode: 9, reward: -389.18, average _reward: -1091.3072284287746
episode: 10, reward: -604.7, average _reward: -1021.0948935301631
episode: 11, reward: -621.59, average _reward: -947.1867884332066
episode: 12, reward: -380.44, average _reward: -877.0021809603483
episode: 13, reward: -129.41, average _reward: -736.1959123258135
episode: 14, reward: -691.51, average _reward: -602.2542054497109
episode: 15, reward: -381.86, average _reward: -547.3192510752684
episode: 16, reward: -128.84, average _reward: -497.30859319638637
episode: 17, reward: -359.89, average _reward: -445.8092512659641
episode: 18, reward: -244.4, average _reward: -419.44159331070597
episode: 19, reward: -256.78, average _reward: -393.18080487241934
episode: 20, reward: -518.37, average _reward: -379.9399186374298
episode: 21, reward: -380.3, average _reward: -371.3067942454835
episode: 22, reward: -508.83, average _reward: -347.17833493811065
episode: 23, reward: -375.96, average _reward: -360.0173593007373
episode: 24, reward: -480.98, average _reward: -384.6719275151437
episode: 25, reward: -268.94, average _reward: -363.6193848574329
episode: 26, reward: -453.77, average _reward: -352.32825458684295
episode: 27, reward: -254.09, average _reward: -384.8210858733029
episode: 28, reward: -639.36, average _reward: -374.24098960333976
episode: 29, reward: -382.74, average _reward: -413.73682978140243
episode: 30, reward: -628.51, average _reward: -426.3329050654291
episode: 31, reward: -379.99, average _reward: -437.3467940545298
episode: 32, reward: -380.14, average _reward: -437.3160179558848
episode: 33, reward: -124.84, average _reward: -424.4473304779398
episode: 34, reward: -359.99, average _reward: -399.3359583314086
episode: 35, reward: -367.32, average _reward: -387.2372003895565
episode: 36, reward: -497.13, average _reward: -397.07491524599584
episode: 37, reward: -241.15, average _reward: -401.41126004706086
episode: 38, reward: -404.69, average _reward: -400.11708876791994
episode: 39, reward: -372.13, average _reward: -376.65042057283983
episode: 40, reward: -146.92, average _reward: -375.59002194970844
episode: 41, reward: -243.92, average _reward: -327.43160526176854
```

Questions assigned to the following page: 3 and 4

```
episode: 42, reward: -506.11, average _reward: -313.82471757723044
episode: 43, reward: -503.0, average _reward: -326.42135441485897
episode: 44, reward: -614.18, average _reward: -364.2369116786662
episode: 45, reward: -248.57, average _reward: -389.65560962901725
episode: 46, reward: -246.32, average _reward: -377.78074944401476
episode: 47, reward: -368.87, average _reward: -352.69955341328773
episode: 48, reward: -401.17, average _reward: -365.4724472967495
episode: 49, reward: -569.02, average _reward: -365.12041862643923
```



Originally the critic should have a higher learning rate than actor, but lets check wgat happens if we set it the opposite way

```
import sys
import gym
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

env = NormalizedEnv(gym.make("Pendulum-v1"))

agent = DDPGagent(env, actor_learning_rate=1e-3,
critic_learning_rate=1e-4,)
noise = OUNoise(env.action_space)
batch_size = 128
```

Question assigned to the following page: 4

```
rewards = []
avg_rewards = []

for episode in range(50):
    state = env.reset()
    noise.reset()
    episode_reward = 0

    for step in range(500):
        action = agent.get_action(state)
        #Add noise to action

        action = noise.get_action(action)
        new_state, reward, done, _ = env.step(action)
        agent.memory.push(state, action, reward, new_state, done)

        if len(agent.memory) > batch_size:
            agent.update(batch_size)

        state = new_state
        episode_reward += reward

        if done:
            sys.stdout.write("episode: {}, reward: {}, average
_reward: {} \n".format(episode, np.round(episode_reward, decimals=2),
np.mean(rewards[-10:])))
            break

    rewards.append(episode_reward)
    avg_rewards.append(np.mean(rewards[-10:]))

plt.plot(rewards)
plt.plot(avg_rewards)
plt.plot()
plt.xlabel('Episode')
plt.ylabel('Reward')
plt.show()
```

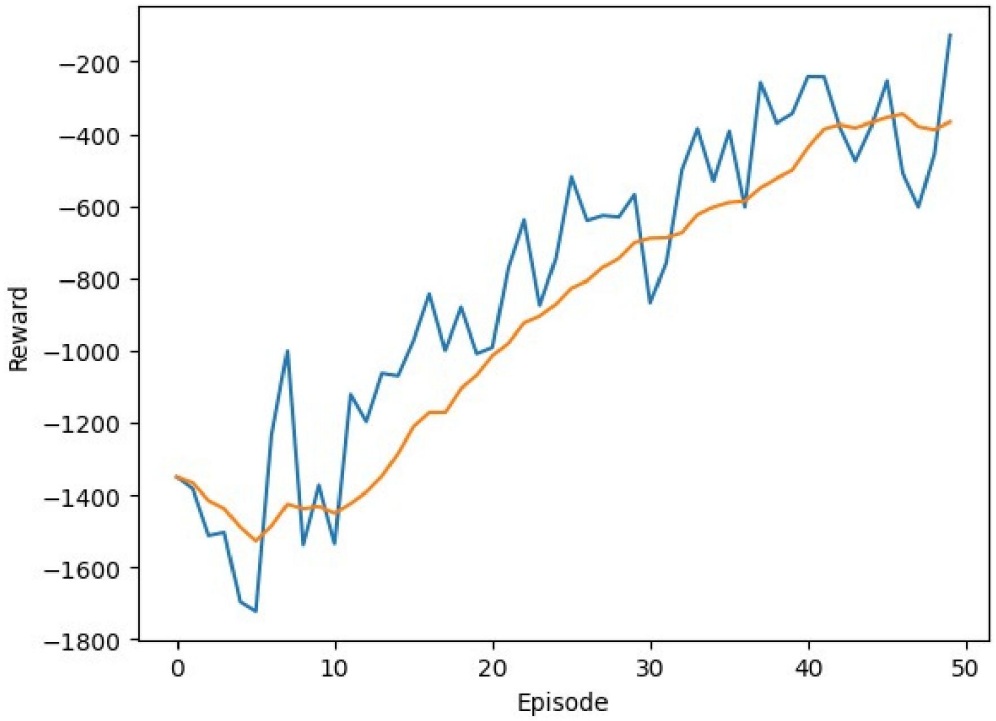
```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to
`transformed_cell` argument and any exception that happen during
thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
/usr/local/lib/python3.10/dist-packages/gym/core.py:317:
DeprecationWarning: WARN: Initializing wrapper in old step API which
returns one bool instead of two. It is recommended to set
`new_step_api=True` to use new step API. This will be the default
behaviour in future.
  deprecation(
```

Question assigned to the following page: 4

```
/usr/local/lib/python3.10/dist-packages/gym/wrappers/step_api_compatib
ility.py:39: DeprecationWarning: WARN: Initializing environment in old
step API which returns one bool instead of two. It is recommended to
set `new_step_api=True` to use new step API. This will be the default
behaviour in future.
  deprecation(

episode: 0, reward: -1349.96, average _reward: nan
episode: 1, reward: -1382.63, average _reward: -1349.9574595246406
episode: 2, reward: -1512.3, average _reward: -1366.2942099671147
episode: 3, reward: -1503.6, average _reward: -1414.9622999629785
episode: 4, reward: -1694.52, average _reward: -1437.1211586186116
episode: 5, reward: -1721.36, average _reward: -1488.6003351593256
episode: 6, reward: -1233.99, average _reward: -1527.3927827154448
episode: 7, reward: -1000.55, average _reward: -1485.4774499514565
episode: 8, reward: -1538.21, average _reward: -1424.8615852832154
episode: 9, reward: -1372.62, average _reward: -1437.4559955895004
episode: 10, reward: -1535.28, average _reward: -1430.972095137761
episode: 11, reward: -1120.36, average _reward: -1449.5044553572602
episode: 12, reward: -1196.57, average _reward: -1423.2778332279643
episode: 13, reward: -1063.04, average _reward: -1391.7050302044438
episode: 14, reward: -1070.13, average _reward: -1347.6495877519073
episode: 15, reward: -970.91, average _reward: -1285.2113540164798
episode: 16, reward: -842.11, average _reward: -1210.1672329809367
episode: 17, reward: -998.89, average _reward: -1170.9792657256398
episode: 18, reward: -879.02, average _reward: -1170.8127804771043
episode: 19, reward: -1008.26, average _reward: -1104.8936744918685
episode: 20, reward: -991.01, average _reward: -1068.458210804211
episode: 21, reward: -771.47, average _reward: -1014.0311542322215
episode: 22, reward: -637.88, average _reward: -979.1417573407874
episode: 23, reward: -874.71, average _reward: -923.2728895323025
episode: 24, reward: -745.85, average _reward: -904.4397098747365
episode: 25, reward: -518.59, average _reward: -872.0108694252424
episode: 26, reward: -640.26, average _reward: -826.7783509536433
episode: 27, reward: -626.15, average _reward: -806.5935316265297
episode: 28, reward: -630.34, average _reward: -769.3199934898505
episode: 29, reward: -567.41, average _reward: -744.4523454323183
episode: 30, reward: -867.72, average _reward: -700.3676014242476
episode: 31, reward: -756.68, average _reward: -688.0387647190688
episode: 32, reward: -500.67, average _reward: -686.5596866405402
episode: 33, reward: -384.48, average _reward: -672.838415907556
episode: 34, reward: -529.84, average _reward: -623.8150347596326
episode: 35, reward: -391.53, average _reward: -602.2148169084021
episode: 36, reward: -602.48, average _reward: -589.5091206189538
episode: 37, reward: -256.97, average _reward: -585.7316644090694
episode: 38, reward: -370.06, average _reward: -548.8133133242447
episode: 39, reward: -343.78, average _reward: -522.785071307899
episode: 40, reward: -240.42, average _reward: -500.4211549260085
episode: 41, reward: -241.07, average _reward: -437.69085350657053
episode: 42, reward: -380.99, average _reward: -386.1294999701836
```

Question assigned to the following page: 4

```
episode: 43, reward: -475.4, average _reward: -374.16158505442024
episode: 44, reward: -380.4, average _reward: -383.25356120703754
episode: 45, reward: -251.97, average _reward: -368.30943620832176
episode: 46, reward: -508.29, average _reward: -354.35345946991157
episode: 47, reward: -602.54, average _reward: -344.9344996726851
episode: 48, reward: -455.33, average _reward: -379.4916861822274
episode: 49, reward: -126.08, average _reward: -388.01850775997616
```



Your Inference

the actor learns too fast compared to the critic, it might start exploiting certain actions prematurely, leading to suboptimal policies. Conversely, if the critic learns too slowly, it might provide inaccurate value estimates, hindering the actor's learning process. This could be seen in the above plot. It also gives a lesser average reward at the end of 49 episodes.

what if both has same learning rate

```
import sys
import gym
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

env = NormalizedEnv(gym.make("Pendulum-v1"))
```

Question assigned to the following page: 4

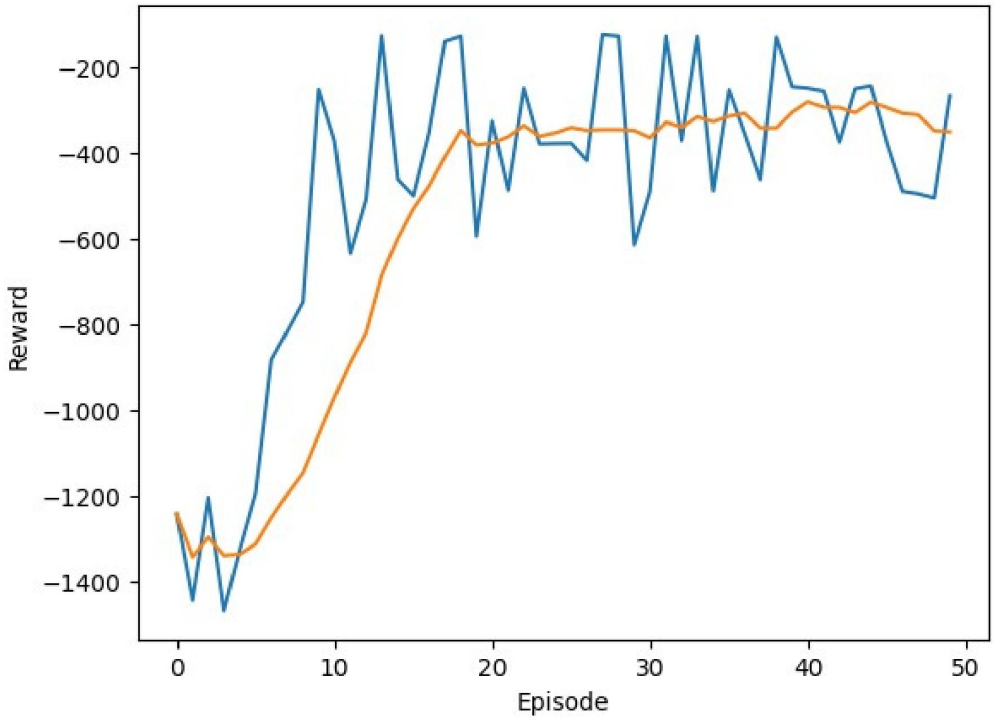
```
agent = DDPGagent(env, actor_learning_rate=1e-3,
critic_learning_rate=1e-3,)
noise = OUNoise(env.action_space)
batch_size = 128
rewards = []
avg_rewards = []

for episode in range(50):
    state = env.reset()
    noise.reset()
    episode_reward = 0

    for step in range(500):
        action = agent.get_action(state)
        #Add noise to action

        action = noise.get_action(action)
        new_state, reward, done, _ = env.step(action)
        agent.memory.push(state, action, reward, new_state, done)

        if len(agent.memory) > batch_size:
            agent.update(batch_size)

        state = new_state
        episode_reward += reward

        if done:
            sys.stdout.write("episode: {}, reward: {}, average
_reward: {} \n".format(episode, np.round(episode_reward, decimals=2),
np.mean(rewards[-10:])))
            break

    rewards.append(episode_reward)
    avg_rewards.append(np.mean(rewards[-10:]))

plt.plot(rewards)
plt.plot(avg_rewards)
plt.plot()
plt.xlabel('Episode')
plt.ylabel('Reward')
plt.show()
```

```
episode: 0, reward: -1243.07, average _reward: nan
episode: 1, reward: -1443.29, average _reward: -1243.0721856638297
episode: 2, reward: -1204.36, average _reward: -1343.1819047518425
episode: 3, reward: -1468.3, average _reward: -1296.9066743013677
episode: 4, reward: -1324.88, average _reward: -1339.7549841103898
episode: 5, reward: -1193.5, average _reward: -1336.7800136416015
episode: 6, reward: -883.09, average _reward: -1312.8994458585896
```

Question assigned to the following page: 4

```
episode: 7, reward: -816.72, average _reward: -1251.498331218494
episode: 8, reward: -748.22, average _reward: -1197.1505761970661
episode: 9, reward: -252.56, average _reward: -1147.2697320952045
episode: 10, reward: -371.98, average _reward: -1057.798498984464
episode: 11, reward: -634.98, average _reward: -970.6891363039724
episode: 12, reward: -509.74, average _reward: -889.8583020795082
episode: 13, reward: -127.24, average _reward: -820.3964558784676
episode: 14, reward: -462.22, average _reward: -686.2909286407227
episode: 15, reward: -500.77, average _reward: -600.0247166742586
episode: 16, reward: -352.87, average _reward: -530.7521415593463
episode: 17, reward: -140.17, average _reward: -477.72970380778526
episode: 18, reward: -128.3, average _reward: -410.0750180093222
episode: 19, reward: -594.78, average _reward: -348.08269885722086
episode: 20, reward: -326.3, average _reward: -382.30496878756355
episode: 21, reward: -487.87, average _reward: -377.73744266943754
episode: 22, reward: -249.66, average _reward: -363.0258765754855
episode: 23, reward: -379.63, average _reward: -337.0178488586069
episode: 24, reward: -378.83, average _reward: -362.2562751892948
episode: 25, reward: -378.42, average _reward: -353.9177417536137
episode: 26, reward: -418.67, average _reward: -341.6830822098235
episode: 27, reward: -124.64, average _reward: -348.26342541292377
episode: 28, reward: -128.23, average _reward: -346.7109766253046
episode: 29, reward: -615.08, average _reward: -346.7041403407476
episode: 30, reward: -490.88, average _reward: -348.7346125028902
episode: 31, reward: -128.04, average _reward: -365.19196633917966
episode: 32, reward: -372.0, average _reward: -329.20923492539464
episode: 33, reward: -128.8, average _reward: -341.4434357963223
episode: 34, reward: -489.07, average _reward: -316.360257123179
episode: 35, reward: -253.69, average _reward: -327.3837488628386
episode: 36, reward: -355.76, average _reward: -314.91063256202926
episode: 37, reward: -463.05, average _reward: -308.61946708524175
episode: 38, reward: -130.56, average _reward: -342.4599262569865
episode: 39, reward: -246.36, average _reward: -342.69258643448245
episode: 40, reward: -249.63, average _reward: -305.8196953506683
episode: 41, reward: -256.77, average _reward: -281.69531481273117
episode: 42, reward: -375.53, average _reward: -294.56820610697673
episode: 43, reward: -251.04, average _reward: -294.9213073191071
episode: 44, reward: -244.44, average _reward: -307.1459863720137
episode: 45, reward: -376.78, average _reward: -282.6831516625783
episode: 46, reward: -490.3, average _reward: -294.99140156339627
episode: 47, reward: -496.02, average _reward: -308.44567216135306
episode: 48, reward: -505.69, average _reward: -311.742877320101
episode: 49, reward: -267.34, average _reward: -349.2562610962955
```

Question assigned to the following page: 4



as seen above This can lead to a more balanced learning process where the actor and critic are synchronized in their updates., also they learn faster as actor now has a higher learning rate

##Changing Tau values

```
import sys
import gym
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

env = NormalizedEnv(gym.make("Pendulum-v1"))

agent = DDPGagent(env, tau = 1)
noise = OUNoise(env.action_space)
batch_size = 128
rewards = []
avg_rewards = []

for episode in range(50):
    state = env.reset()
    noise.reset()
    episode_reward = 0

    for step in range(500):
```

Question assigned to the following page: 4

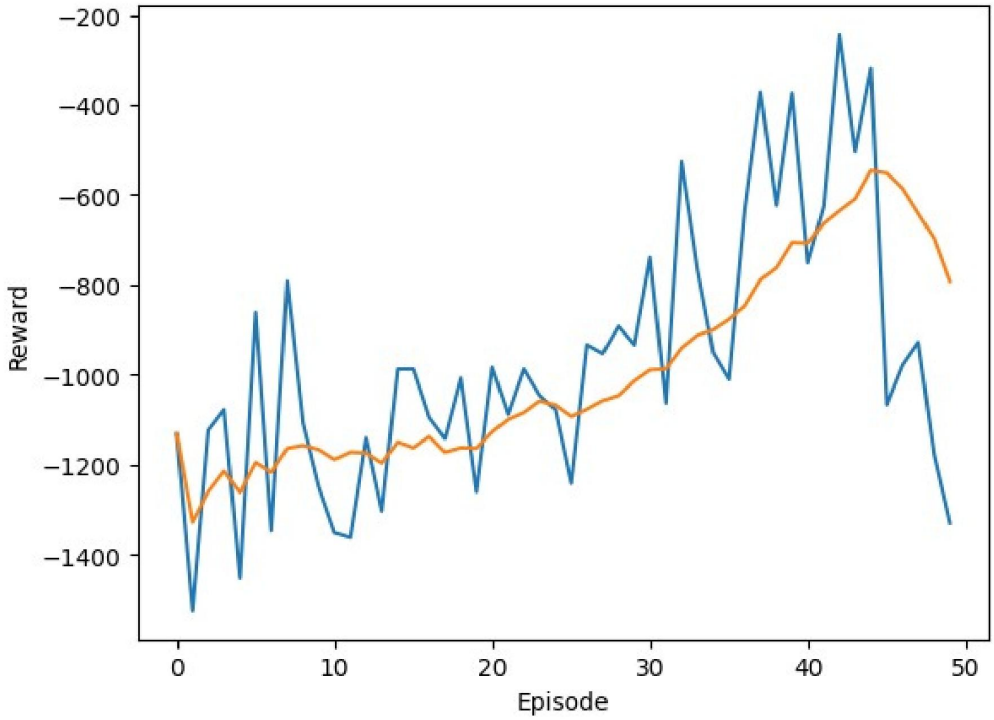
```
        action = agent.get_action(state)
        #Add noise to action

        action = noise.get_action(action)
        new_state, reward, done, _ = env.step(action)
        agent.memory.push(state, action, reward, new_state, done)

        if len(agent.memory) > batch_size:
            agent.update(batch_size)

        state = new_state
        episode_reward += reward

        if done:
            sys.stdout.write("episode: {}, reward: {}, average
_reward: {} \n".format(episode, np.round(episode_reward, decimals=2),
np.mean(rewards[-10:])))
            break

    rewards.append(episode_reward)
    avg_rewards.append(np.mean(rewards[-10:]))

plt.plot(rewards)
plt.plot(avg_rewards)
plt.plot()
plt.xlabel('Episode')
plt.ylabel('Reward')
plt.show()
```

```
episode: 0, reward: -1130.74, average _reward: nan
episode: 1, reward: -1523.72, average _reward: -1130.7395941649968
episode: 2, reward: -1122.82, average _reward: -1327.230801593827
episode: 3, reward: -1076.93, average _reward: -1259.0946195259612
episode: 4, reward: -1451.61, average _reward: -1213.5541781396291
episode: 5, reward: -861.09, average _reward: -1261.1650202787282
episode: 6, reward: -1346.16, average _reward: -1194.4865988081624
episode: 7, reward: -791.44, average _reward: -1216.154916446969
episode: 8, reward: -1106.16, average _reward: -1163.0655150251068
episode: 9, reward: -1248.27, average _reward: -1156.7427291874562
episode: 10, reward: -1350.59, average _reward: -1165.8957841976808
episode: 11, reward: -1361.28, average _reward: -1187.881206943483
episode: 12, reward: -1139.66, average _reward: -1171.6369972577036
episode: 13, reward: -1302.52, average _reward: -1173.321145267996
episode: 14, reward: -986.86, average _reward: -1195.8799925446833
episode: 15, reward: -986.84, average _reward: -1149.4050529615236
episode: 16, reward: -1094.26, average _reward: -1161.9794056619667
episode: 17, reward: -1141.24, average _reward: -1136.7885940451988
episode: 18, reward: -1006.69, average _reward: -1171.7690486440958
episode: 19, reward: -1259.9, average _reward: -1161.8223557544766
episode: 20, reward: -982.86, average _reward: -1162.9849881619637
```

Question assigned to the following page: 4

```
episode: 21, reward: -1087.83, average _reward: -1126.2118693180432
episode: 22, reward: -986.6, average _reward: -1098.8669109541302
episode: 23, reward: -1045.63, average _reward: -1083.5605768473615
episode: 24, reward: -1076.91, average _reward: -1057.8714996015365
episode: 25, reward: -1240.81, average _reward: -1066.8763642442657
episode: 26, reward: -933.02, average _reward: -1092.2740112028928
episode: 27, reward: -952.41, average _reward: -1076.150816181284
episode: 28, reward: -891.75, average _reward: -1057.2674936978426
episode: 29, reward: -933.41, average _reward: -1045.7730562620811
episode: 30, reward: -738.46, average _reward: -1013.1237766692741
episode: 31, reward: -1062.57, average _reward: -988.6837044438741
episode: 32, reward: -525.87, average _reward: -986.1577906654193
episode: 33, reward: -764.59, average _reward: -940.0846278491848
episode: 34, reward: -950.48, average _reward: -911.9800856147413
episode: 35, reward: -1009.87, average _reward: -899.3375265026159
episode: 36, reward: -635.61, average _reward: -876.2428879828154
episode: 37, reward: -371.76, average _reward: -846.5010888034324
episode: 38, reward: -623.3, average _reward: -788.4356558858816
episode: 39, reward: -373.78, average _reward: -761.5904289852282
episode: 40, reward: -750.91, average _reward: -705.627253080355
episode: 41, reward: -625.31, average _reward: -706.8718641584408
episode: 42, reward: -243.49, average _reward: -663.145930121904
episode: 43, reward: -503.66, average _reward: -634.9085138240569
episode: 44, reward: -319.1, average _reward: -608.8158195037511
episode: 45, reward: -1066.34, average _reward: -545.6780324039607
episode: 46, reward: -977.92, average _reward: -551.325303068983
episode: 47, reward: -927.38, average _reward: -585.557049974173
episode: 48, reward: -1176.82, average _reward: -641.1191449730775
episode: 49, reward: -1328.87, average _reward: -696.4719469855911
```

Question assigned to the following page: 4



A higher tau value means a higher exploration, so it needs more episodes to learn, thats why it has lesser rewards, as shown in the plot above at the end of 50 episodes.

```
import sys
import gym
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

env = NormalizedEnv(gym.make("Pendulum-v1"))

agent = DDPGagent(env, tau = 0.00001)
noise = OUNoise(env.action_space)
batch_size = 128
rewards = []
avg_rewards = []

for episode in range(50):
    state = env.reset()
    noise.reset()
    episode_reward = 0

    for step in range(500):
        action = agent.get_action(state)
        #Add noise to action
```

Question assigned to the following page: 4

```
        action = noise.get_action(action)
        new_state, reward, done, _ = env.step(action)
        agent.memory.push(state, action, reward, new_state, done)

        if len(agent.memory) > batch_size:
            agent.update(batch_size)

        state = new_state
        episode_reward += reward

        if done:
            sys.stdout.write("episode: {}, reward: {}, average
_reward: {} \n".format(episode, np.round(episode_reward, decimals=2),
np.mean(rewards[-10:])))
            break

    rewards.append(episode_reward)
    avg_rewards.append(np.mean(rewards[-10:]))

plt.plot(rewards)
plt.plot(avg_rewards)
plt.plot()
plt.xlabel('Episode')
plt.ylabel('Reward')
plt.show()
```

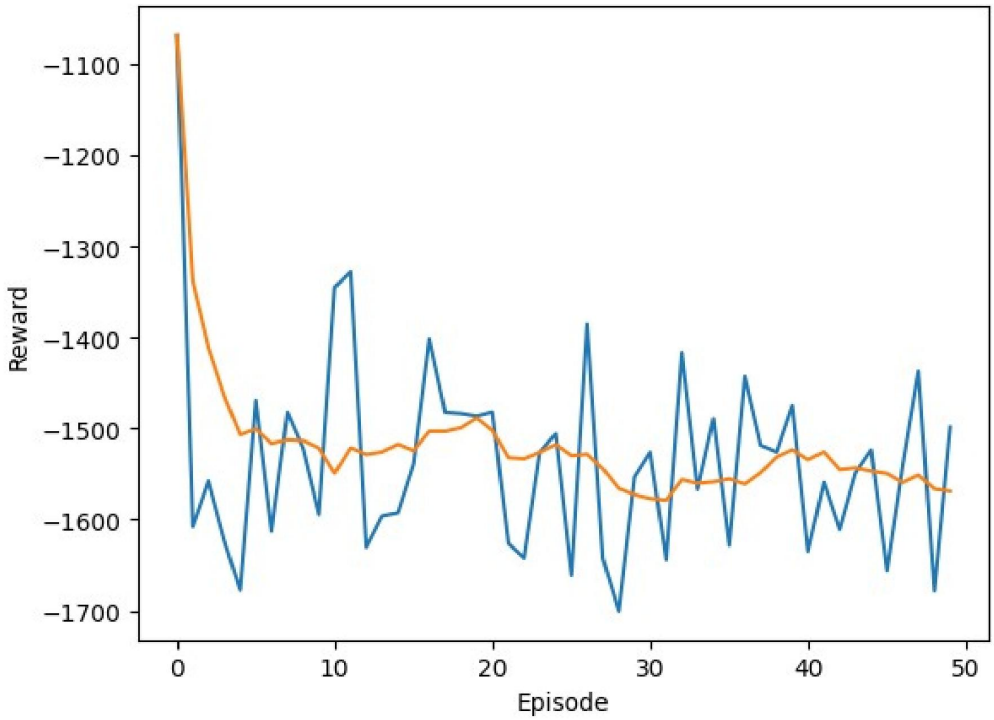
```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to
`transformed_cell` argument and any exception that happen during
thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
/usr/local/lib/python3.10/dist-packages/gym/core.py:317:
DeprecationWarning: WARN: Initializing wrapper in old step API which
returns one bool instead of two. It is recommended to set
`new_step_api=True` to use new step API. This will be the default
behaviour in future.
  deprecation(
/usr/local/lib/python3.10/dist-packages/gym/wrappers/step_api_compatib
ility.py:39: DeprecationWarning: WARN: Initializing environment in old
step API which returns one bool instead of two. It is recommended to
set `new_step_api=True` to use new step API. This will be the default
behaviour in future.
  deprecation(
/usr/local/lib/python3.10/dist-packages/gym/utils/passive_env_checker.
py:241: DeprecationWarning: `np.bool8` is a deprecated alias for
`np.bool_`.  (Deprecated NumPy 1.24)
  if not isinstance(terminated, (bool, np.bool8)):
/usr/local/lib/python3.10/dist-packages/numpy/core/fromnumeric.py:3504
```

Question assigned to the following page: 4

```
: RuntimeWarning: Mean of empty slice.
  return _methods._mean(a, axis=axis, dtype=dtype,
/usr/local/lib/python3.10/dist-packages/numpy/core/_methods.py:129:
RuntimeWarning: invalid value encountered in scalar divide
  ret = ret.dtype.type(ret / rcount)

episode: 0, reward: -1067.89, average _reward: nan
episode: 1, reward: -1607.81, average _reward: -1067.8890907377265
episode: 2, reward: -1557.61, average _reward: -1337.8514564751904
episode: 3, reward: -1622.79, average _reward: -1411.1054805576714
episode: 4, reward: -1677.6, average _reward: -1464.027430033484
episode: 5, reward: -1468.99, average _reward: -1506.7419270834048
episode: 6, reward: -1612.87, average _reward: -1500.4507036985672
episode: 7, reward: -1482.21, average _reward: -1516.51057400468
episode: 8, reward: -1521.99, average _reward: -1512.222427241261
episode: 9, reward: -1594.4, average _reward: -1513.3077460938946
episode: 10, reward: -1344.82, average _reward: -1521.41701830745
episode: 11, reward: -1327.87, average _reward: -1549.1097077147047
episode: 12, reward: -1630.45, average _reward: -1521.115278244632
episode: 13, reward: -1596.12, average _reward: -1528.3990183491849
episode: 14, reward: -1592.52, average _reward: -1525.73160984345
episode: 15, reward: -1538.64, average _reward: -1517.2231566266028
episode: 16, reward: -1401.25, average _reward: -1524.1881699389592
episode: 17, reward: -1482.32, average _reward: -1503.0265637391833
episode: 18, reward: -1483.52, average _reward: -1503.038412055093
episode: 19, reward: -1486.34, average _reward: -1499.191568392785
episode: 20, reward: -1481.95, average _reward: -1488.385584238133
episode: 21, reward: -1626.22, average _reward: -1502.0992220781718
episode: 22, reward: -1642.1, average _reward: -1531.9338070987467
episode: 23, reward: -1526.08, average _reward: -1533.0983457803886
episode: 24, reward: -1505.85, average _reward: -1526.0942689418862
episode: 25, reward: -1661.16, average _reward: -1517.4274410726005
episode: 26, reward: -1385.45, average _reward: -1529.678894969195
episode: 27, reward: -1642.6, average _reward: -1528.0983801907655
episode: 28, reward: -1700.62, average _reward: -1544.1260479711232
episode: 29, reward: -1553.29, average _reward: -1565.835867885618
episode: 30, reward: -1525.85, average _reward: -1572.5307675908673
episode: 31, reward: -1644.02, average _reward: -1576.92013936729
episode: 32, reward: -1416.56, average _reward: -1578.700524358223
episode: 33, reward: -1566.85, average _reward: -1556.1472138301037
episode: 34, reward: -1489.23, average _reward: -1560.224814329213
episode: 35, reward: -1628.31, average _reward: -1558.563058286511
episode: 36, reward: -1442.69, average _reward: -1555.2781882659615
episode: 37, reward: -1518.25, average _reward: -1561.0021081914037
episode: 38, reward: -1525.77, average _reward: -1548.566837686543
episode: 39, reward: -1474.45, average _reward: -1531.0818431209714
episode: 40, reward: -1634.81, average _reward: -1523.1976016746314
episode: 41, reward: -1559.0, average _reward: -1534.0944149880254
episode: 42, reward: -1610.96, average _reward: -1525.5926155764694
episode: 43, reward: -1548.91, average _reward: -1545.032470663244
```

Question assigned to the following page: 4

```
episode: 44, reward: -1523.42, average _reward: -1543.2383298779982
episode: 45, reward: -1656.21, average _reward: -1546.6576223507795
episode: 46, reward: -1540.61, average _reward: -1549.4474729802087
episode: 47, reward: -1437.04, average _reward: -1559.239424215881
episode: 48, reward: -1678.35, average _reward: -1551.1186652847239
episode: 49, reward: -1498.53, average _reward: -1566.3770120071454
```



A very lesser tau value shows poor training, leading to suboptimal solution due to lack of exploration and too much exploitation