Question 1

Q 1(a)

MountainCar V0

Observations
State Space: The state space consists of two continuous variables:

Position of the car (ranging from -1.2 to 0.6)
Velocity of the car (ranging from -0.07 to 0.07)
Action Space: The action space is discrete with three possible actions:

0: Push the car left
1: Do nothing
2: Push the car right

Reward: For each time step, the agent receives a reward of -1 until it reaches the goal at the top of the mountain on the right (position >= 0.5), at which point the episode terminates. The constant negative reward encourages the agent to reach the goal as quickly as possible to minimize the number of time steps.

Objective: The random agent does not learn or optimize behavior, so it typically takes a high number of steps without reaching the goal due to random action selection, highlighting the need for a more intelligent policy for success.

PongV0
Observations

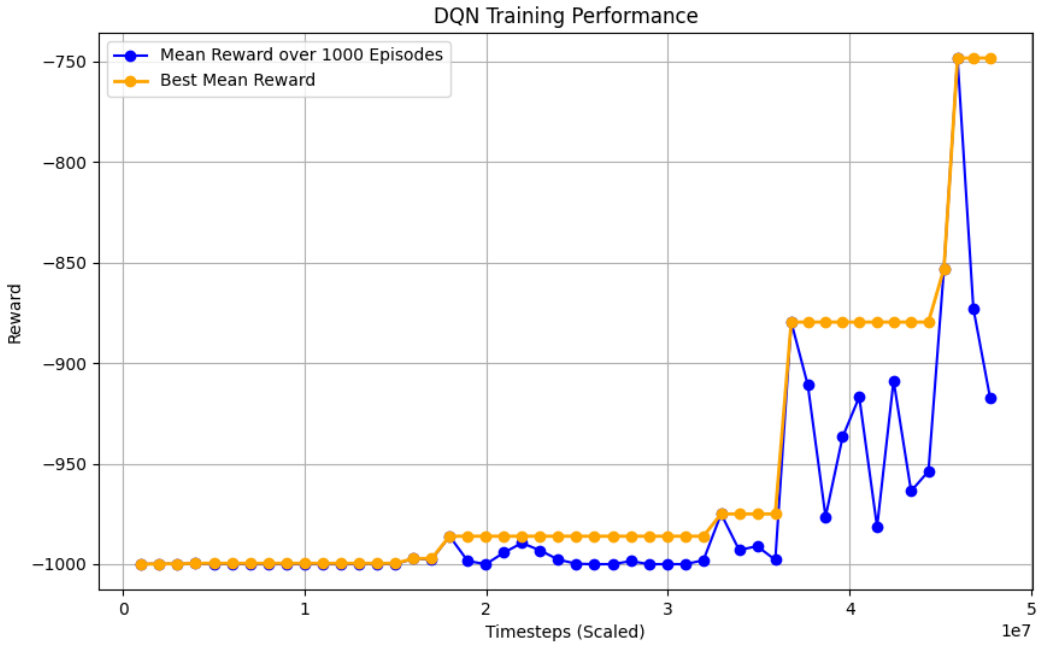
State and Action Space:
The state space consists of frames (pixel observations of size (210, 160, 3) in RGB).
The action space is discrete, representing various actions the paddle can take (like moving up, down, or no movement).

Random Agent:
The random agent selects actions using env.action_space.sample(), simulating random movements of the paddle.
The loop runs until the episode ends (done becomes True).

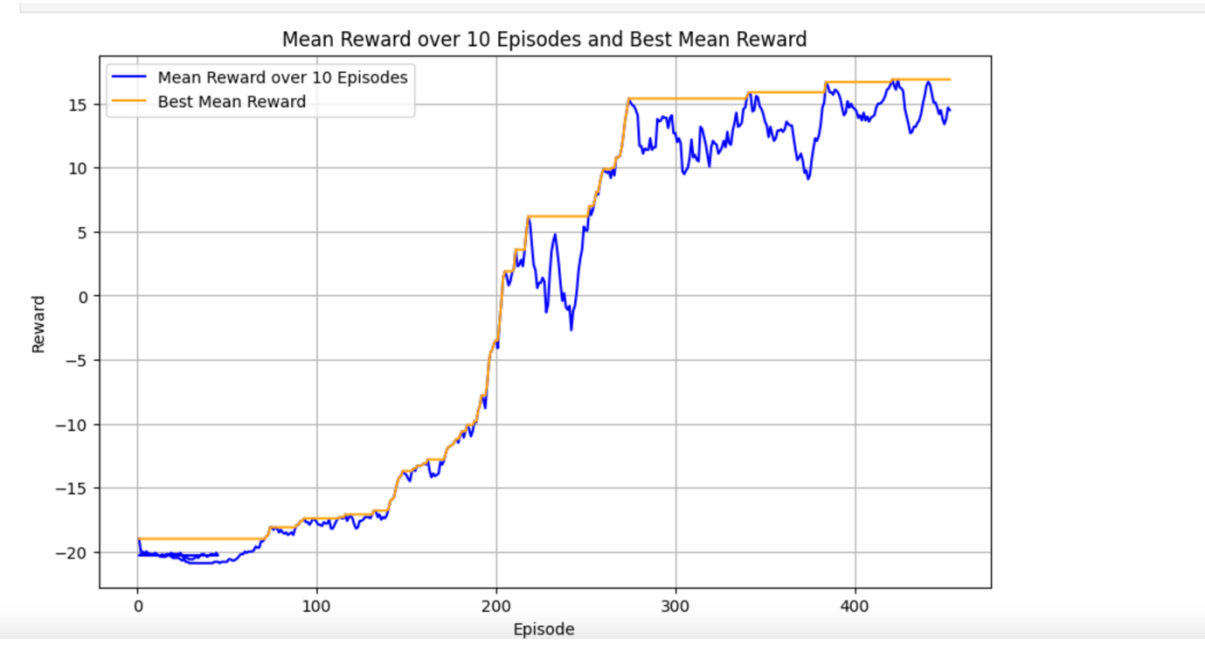
Observing Rewards:
The code prints the state shape, actions, and rewards at every 1000 steps to avoid clutter. You can adjust this value as needed.
The total reward and number of steps are accumulated.

Q2(b)

Mountain Car V0



PongV0



Observations

1. The graph of Mean reward with a suitable window size and time steps has been plotted for MountainCarV0 and PongV0
2. We observe that with increase in timesteps/episodes the mean n episode reward approaches and converges with the best mean reward for both the graphs.

Observations

Above is the graph that explains the action choices of the trained agent for various values of position and velocity

With the most optimized model trained we get the above graph showing us

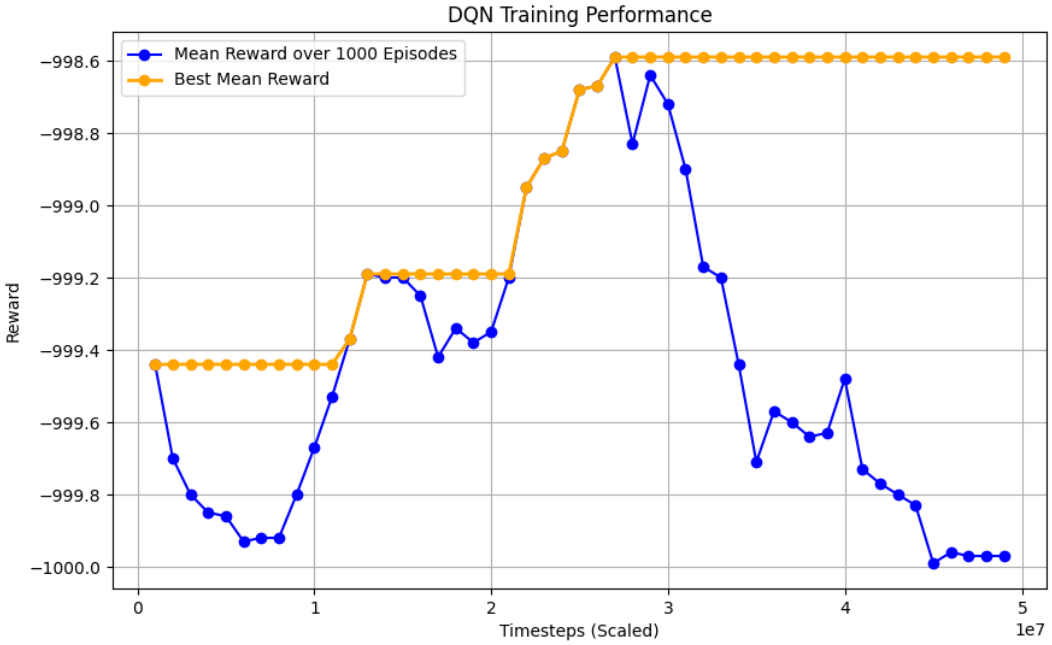
1. 1. For Positions>0 The most probable action taken is accelerate right
2. 2. For Position<0 primarily with higher negative values the car is expected to accelerate left

Note: The code for doing pre-processing is included in corresponding jupyter notebooks and python scripts for both environments respectively.

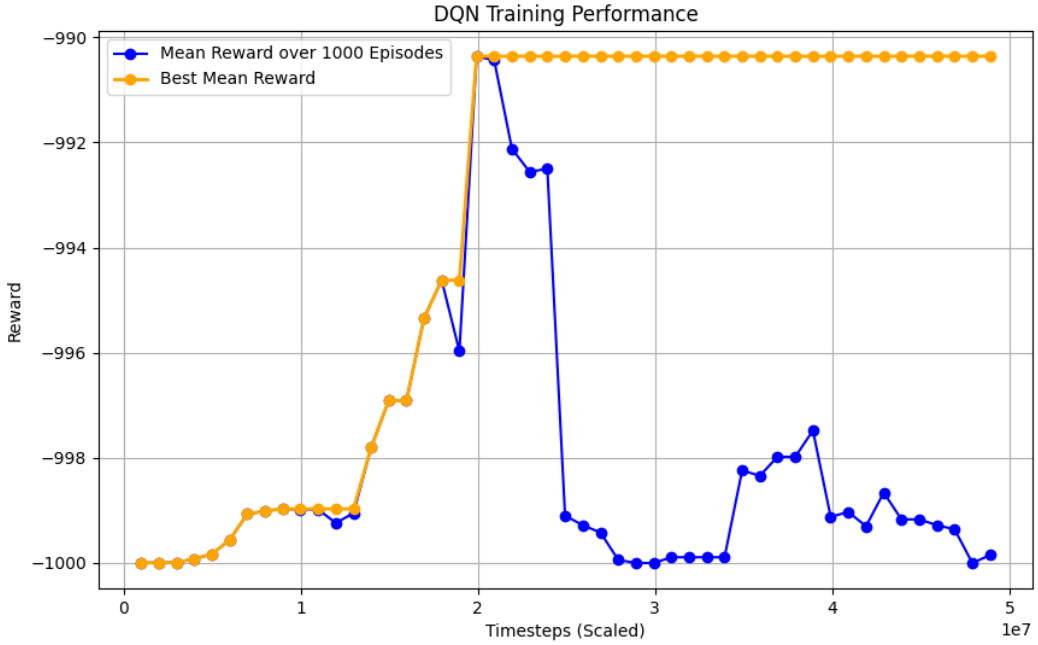
Q1 (c).

I generate the graphs choosing various variations of the learning_rate hyperparameter with values 0.1,0.01,0.001 respectively.

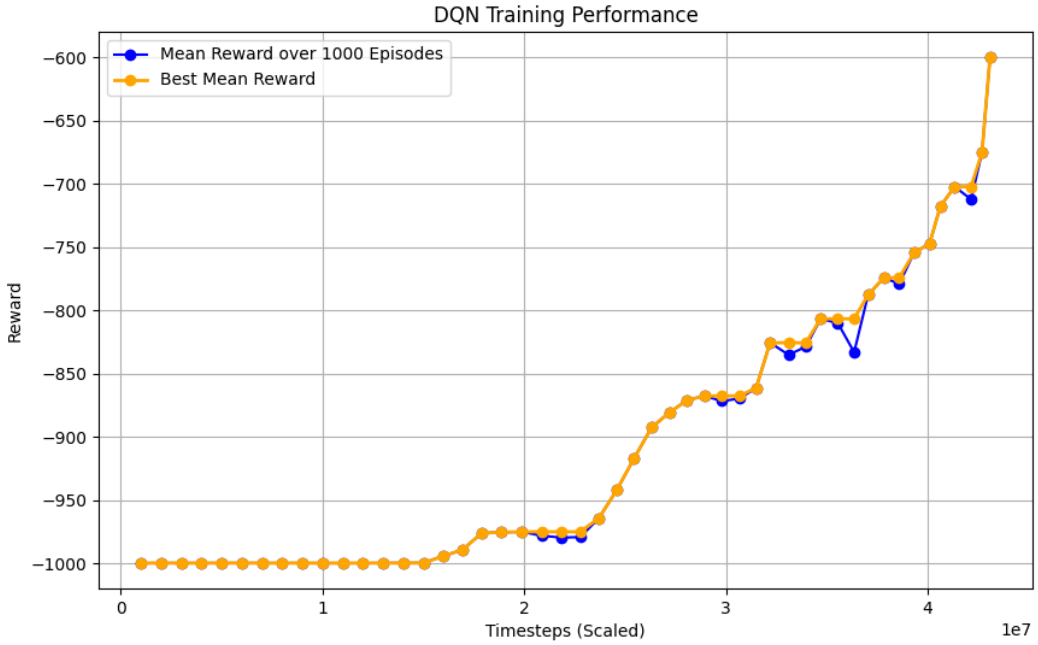
LearningRate=0.1

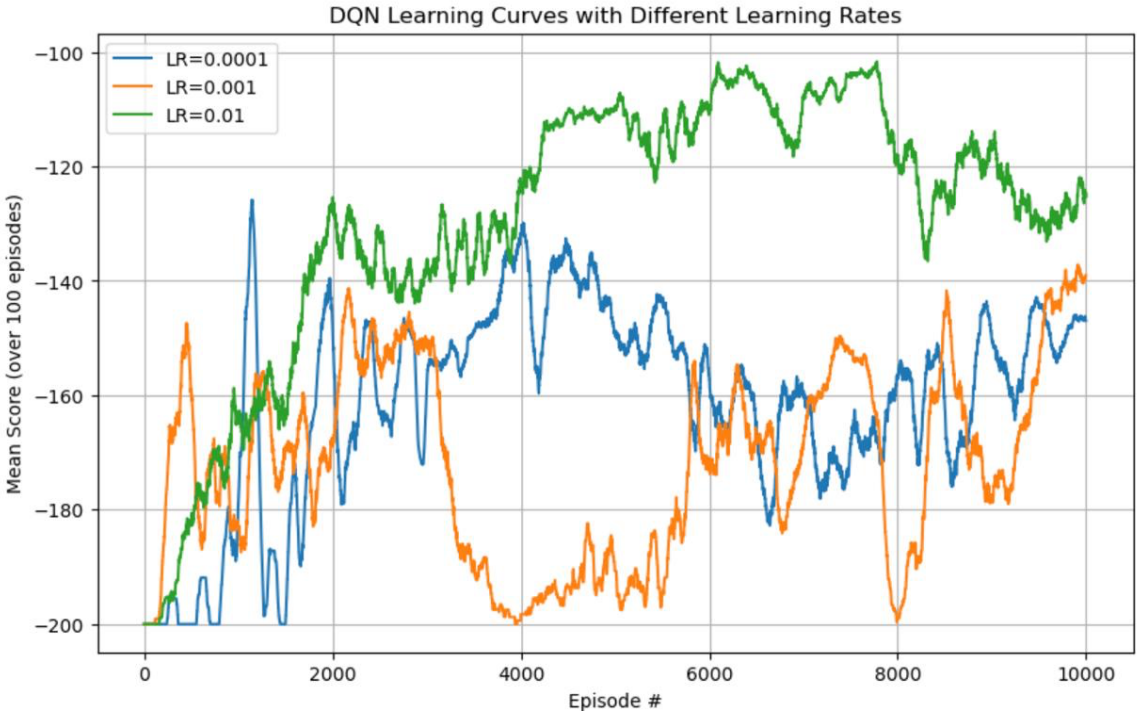


Learning Rate=0.01



Learning Rate=0.001





Observations

1. I observe that with 0.1 learning rates the fluctuations is very high with no convergence at all

2. With 0.01 learning rates I observe the fluctuations in producing the score to be comparatively less but still the values don't converge properly.
3. With 0.001 learning rate the the n step mean of reward and best mean reward converges and n step mean reward follows best mean reward with increase in timesteps.

QUESTION 2

Note:

1. The python script is q2.py is present to execute the script through command line
2. The Jupyter notebook is also present having the solution to all subparts and all experiments carried out

2(a)
Observations -

LunarLander V2

The state space of LunarLander is an 8-dimensional continuous space: [x position, y position, x velocity, y velocity, angle, angular velocity, left leg contact, right leg contact].

The action space is discrete with four possible actions:
0 = Do nothing
1 = Fire left engine
2 = Fire main engine
3 = Fire right engine

Random Agent
This agent samples actions randomly from the action space and executes them, which means it has no control strategy.

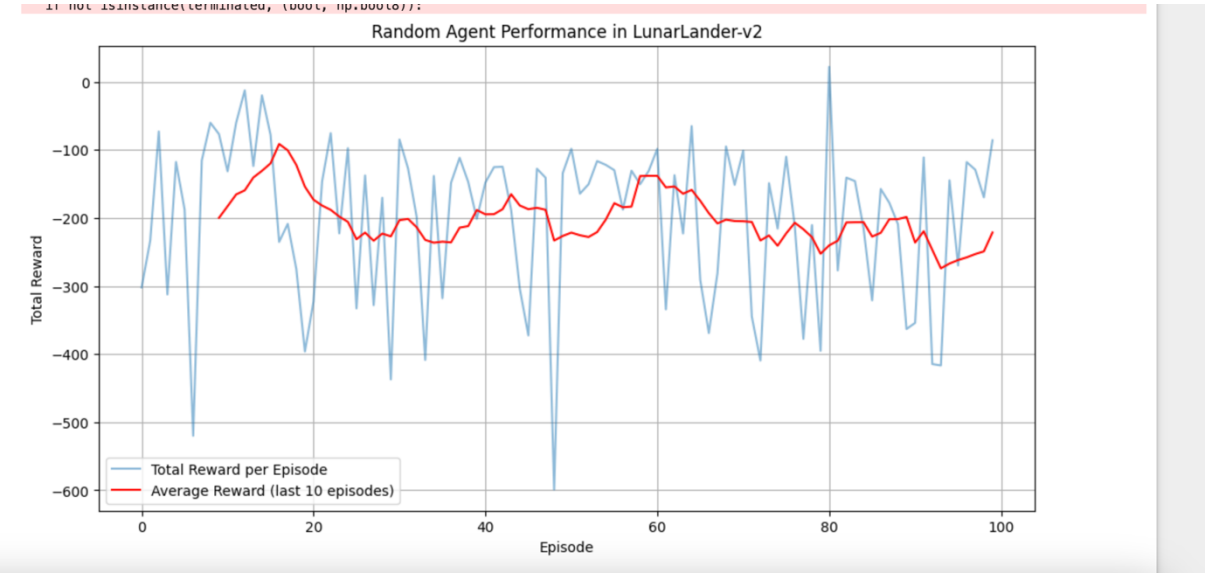
Reward: The LunarLander environment provides rewards based on specific goals and penalties:

Reward of +100 to +140 for successfully landing.
Reward is higher for landing with both legs touching down.
Penalty for crashing (large negative reward).
Small reward/penalty adjustments based on proximity to the landing pad, velocity, angle, and fuel usage.

Behavior of Random Agent :

The random agent typically ends up crashing, resulting in large negative rewards. Occasionally, it may achieve minor positive rewards by chance when it stays closer to the center or slows its descent.

The total reward varies widely, usually ending up negative, as random actions generally lead to an unstable descent or crash



1(a).2 CartPoleV0 with Random Agent

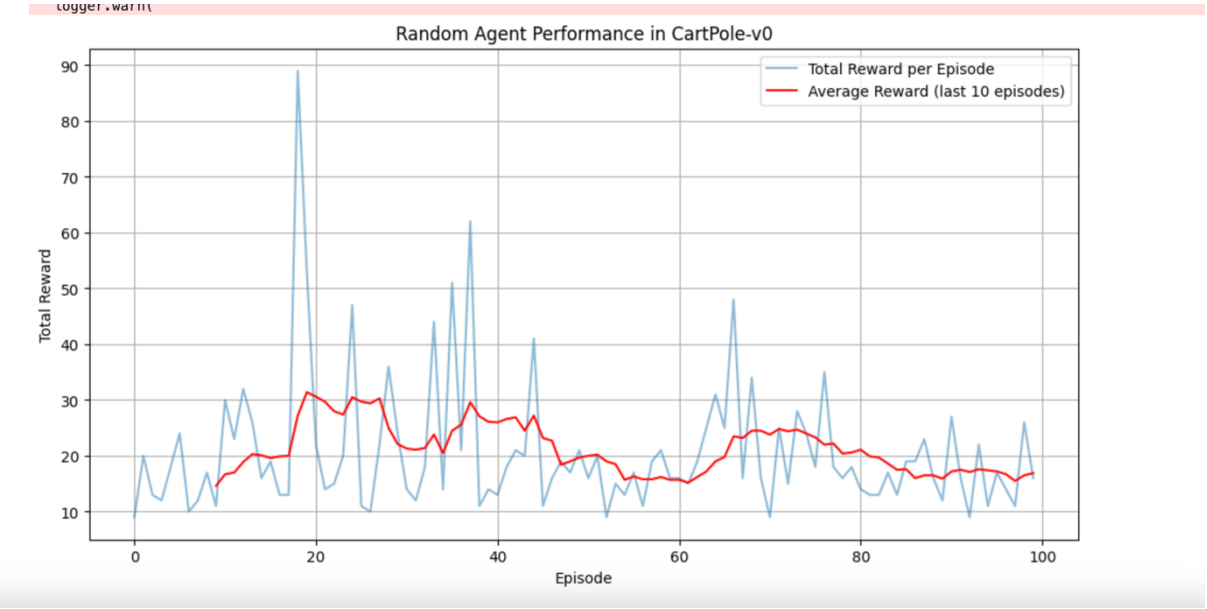
Observations

CartPole V0
The state space of CartPole is a 4-dimensional continuous space: [position of cart, velocity of cart, angle of pole, rotation rate of pole].
The **action space** is discrete with two possible actions:
0 = Move the cart left
1 = Move the cart right
The CartPole environment provides a reward of 1 for each time step the pole remains balanced.

Random Agent:
The random agent samples actions from the action space and executes them without regard to the current state.

Reward:
In CartPole, each time step in which the pole remains upright gives a reward of 1. The episode ends when the pole falls or the cart moves too far from the center.

Since the agent is acting randomly, it tends to balance the pole only briefly, usually leading to low rewards (between 10 and 50 in most cases, but varies
High rewards (close to the maximum of 200) would require consistent corrective actions, which random sampling rarely achieves.

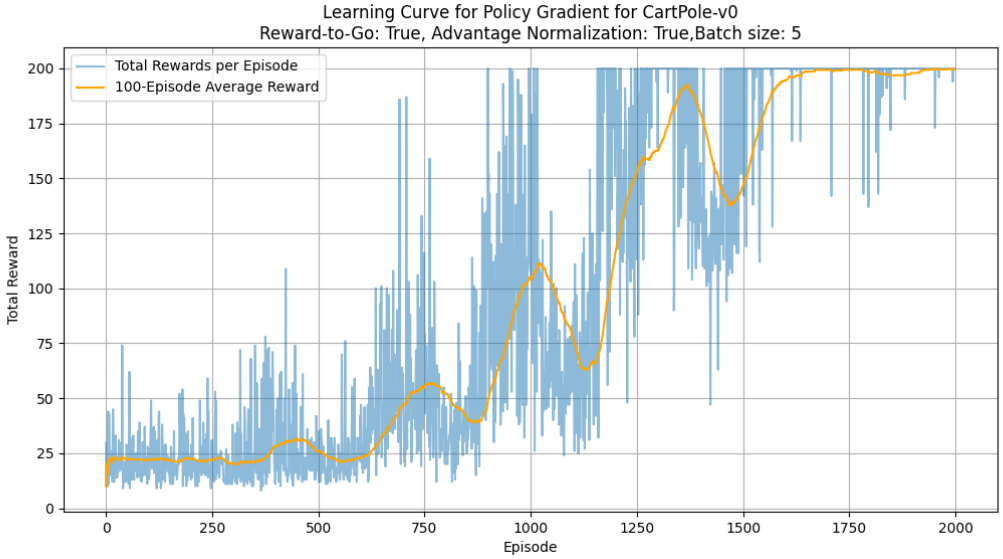


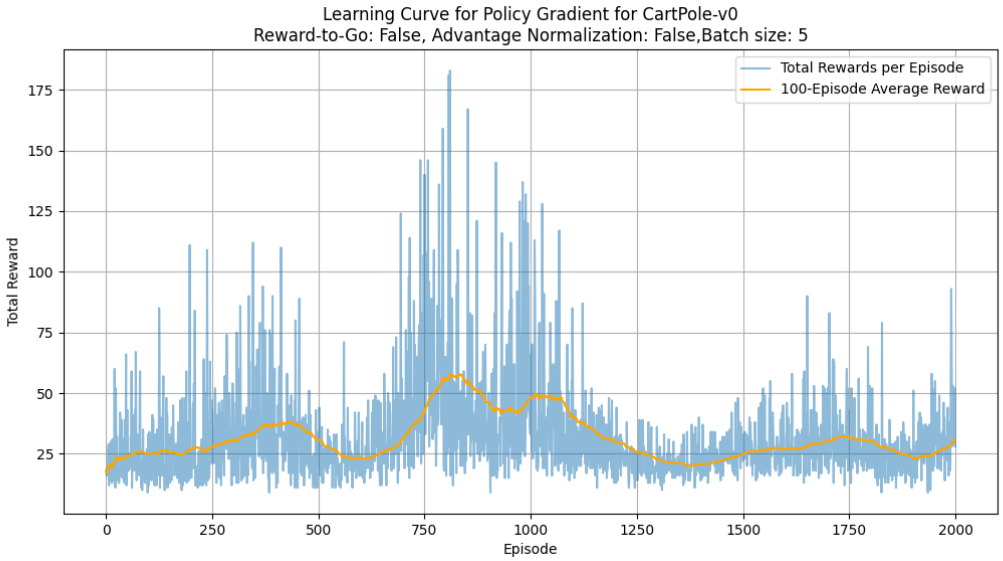
Q2(b)

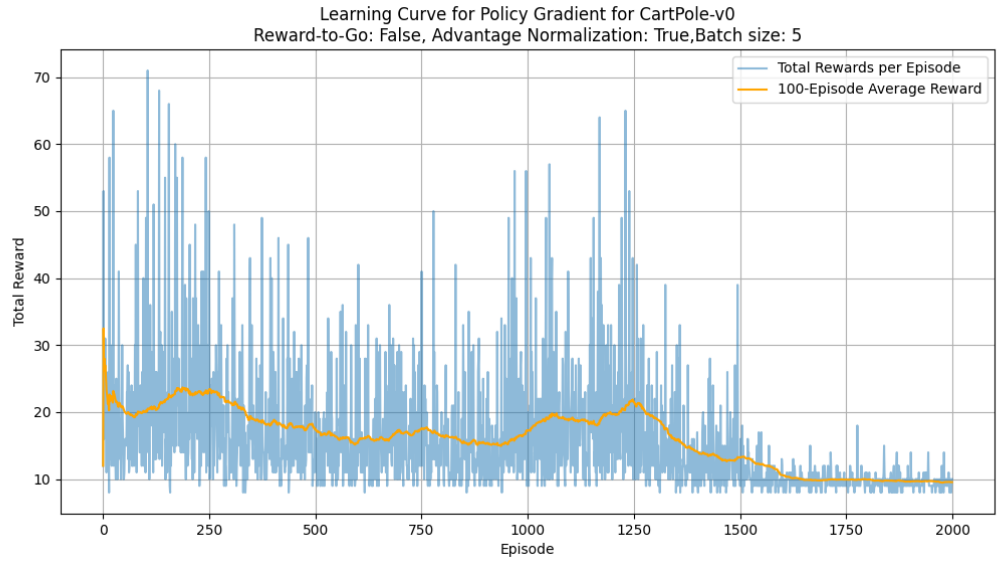
CartPoleV0

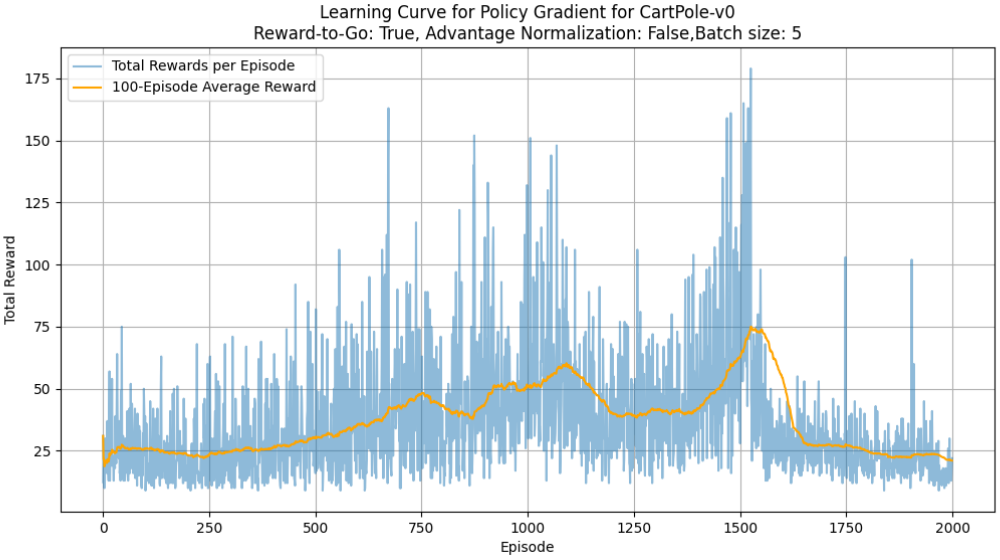
Parameters

gamma = 0.99
reward_to_go=True
advantage_normalization=True
env_name = "CartPole-v0"
baseline_type = "time-dependent"
batch_size = 5
num_games = 2000
learning_rate = 0.001

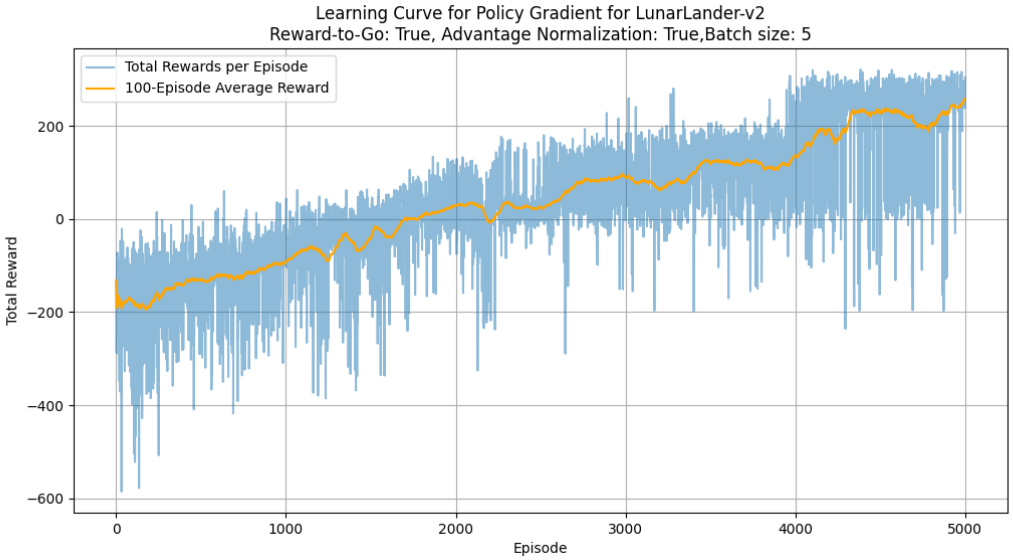

Learning Curve for Policy Gradient for CartPole-v0
Reward-to-Go: True, Advantage Normalization: True,Batch size: 5
Total Rewards per Episode
100-Episode Average Reward
Total Reward
Episode
0
25
50
75
100
125
150
175
200
0
250
500
750
1000
1250
1500
1750
2000

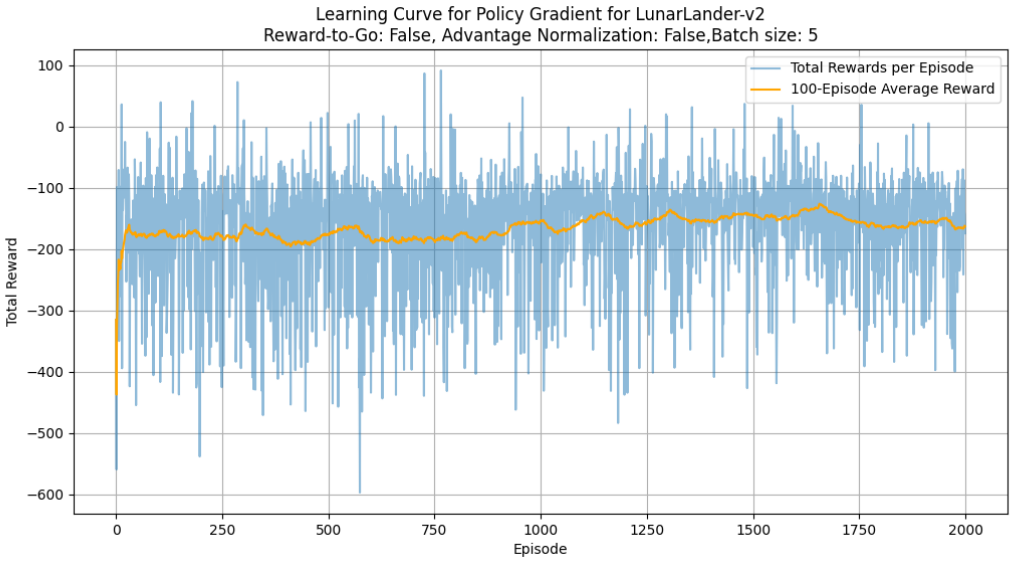

Learning Curve for Policy Gradient for CartPole-v0
Reward-to-Go: False, Advantage Normalization: False,Batch size: 5
Total Rewards per Episode
100-Episode Average Reward
Total Reward
Episode
25
50
75
100
125
150
175
0
250
500
750
1000
1250
1500
1750
2000

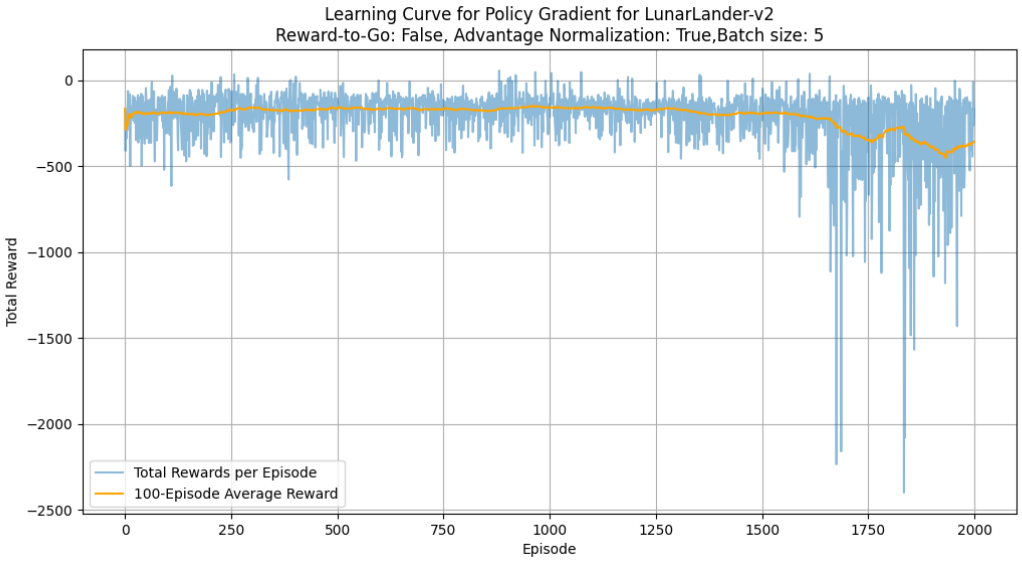

Learning Curve for Policy Gradient for CartPole-v0
Reward-to-Go: False, Advantage Normalization: True,Batch size: 5
Total Rewards per Episode
100-Episode Average Reward
Total Reward
Episode
10
20
30
40
50
60
70
0
250
500
750
1000
1250
1500
1750
2000

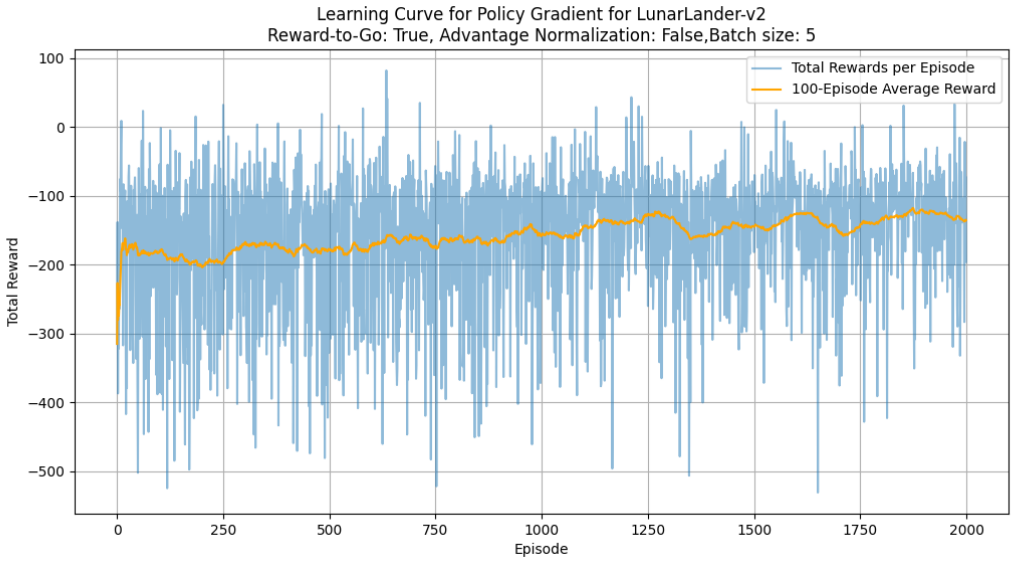

Learning Curve for Policy Gradient for CartPole-v0
Reward-to-Go: True, Advantage Normalization: False,Batch size: 5
Total Rewards per Episode
100-Episode Average Reward
Total Reward
Episode
25
50
75
100
125
150
175
0
250
500
750
1000
1250
1500
1750
2000

LunarLanderV2









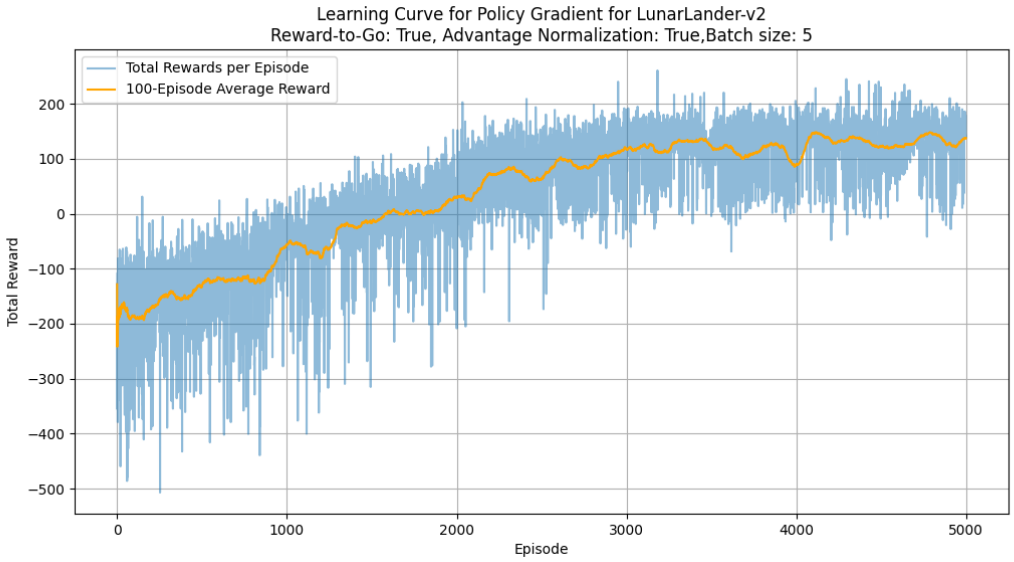
Observation

We observe the in case of advantage normalization=True, and Reward To Go functionality to be True the convergence is achieved.
In other cases there is presence of high variance and not giving fetching good rewards.

Lunar Lander

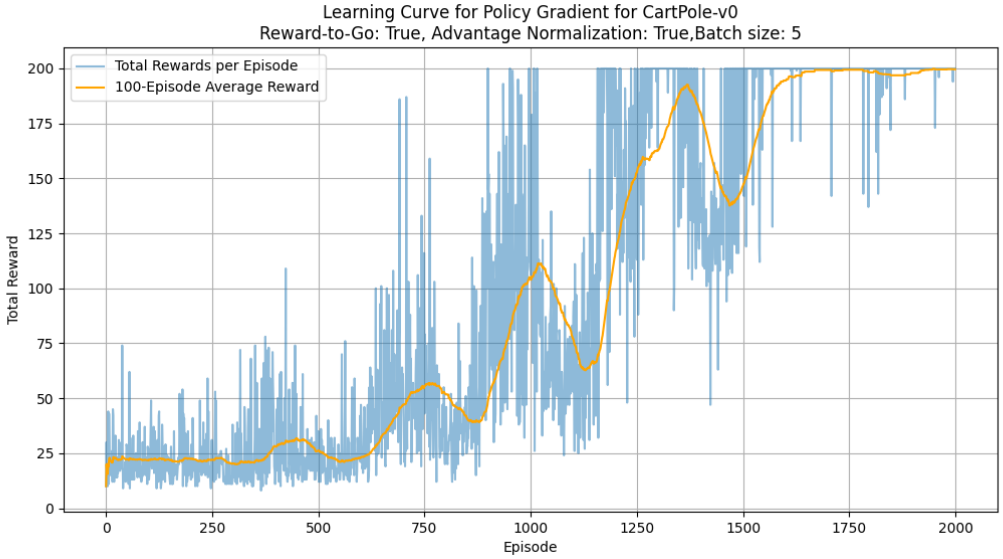
# Parameters
gamma = 0.99

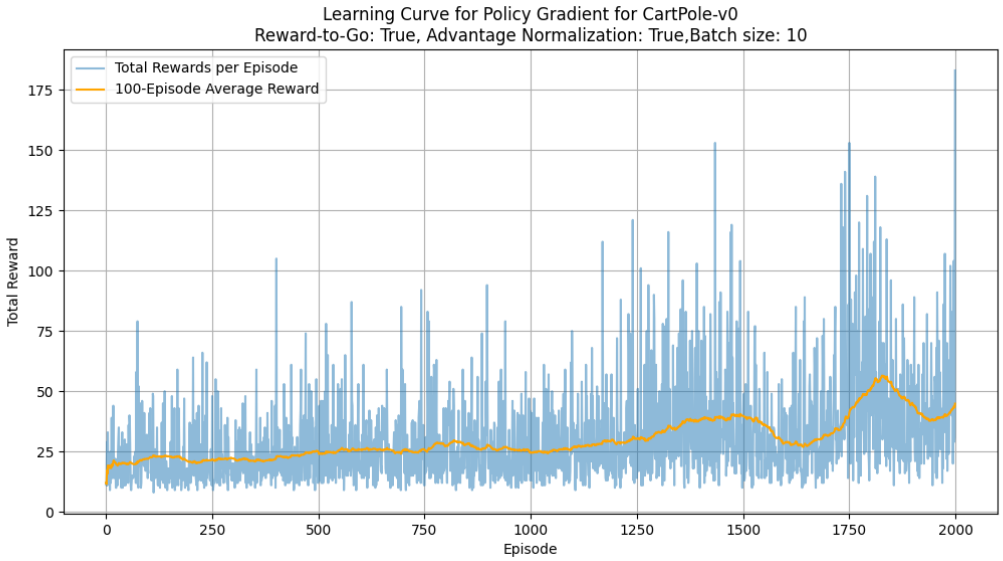
mode = "train"
env_name = "LunarLander-v2" # to prepare the results for Lunar Lander
reward_to_go=True
advantage_normalization=True
baseline_type = "time-dependent"
batch_size = 5 # Number of episodes per batch
env_max_num_steps = 1000
 num_games = 5000
  learning_rate = 0.0005

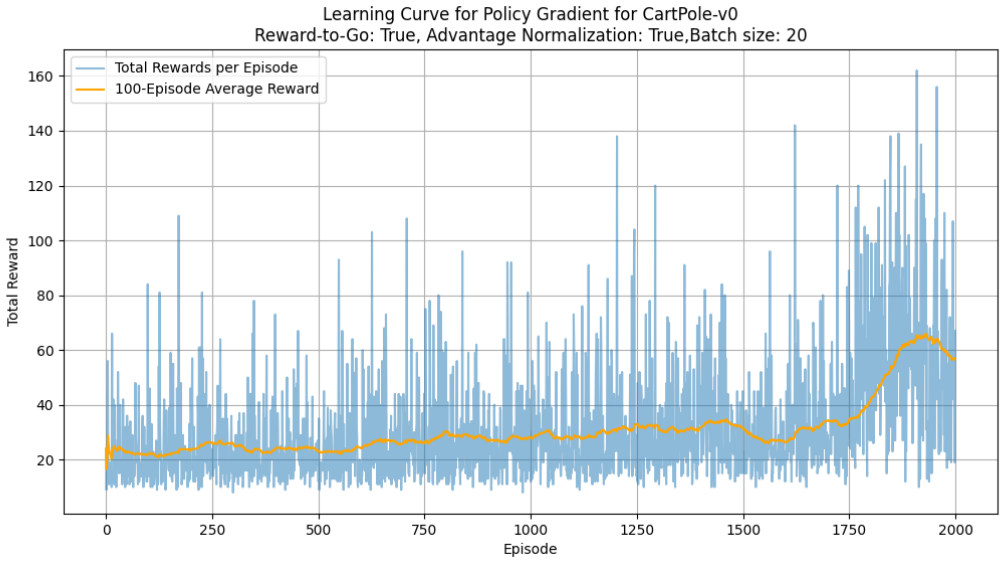


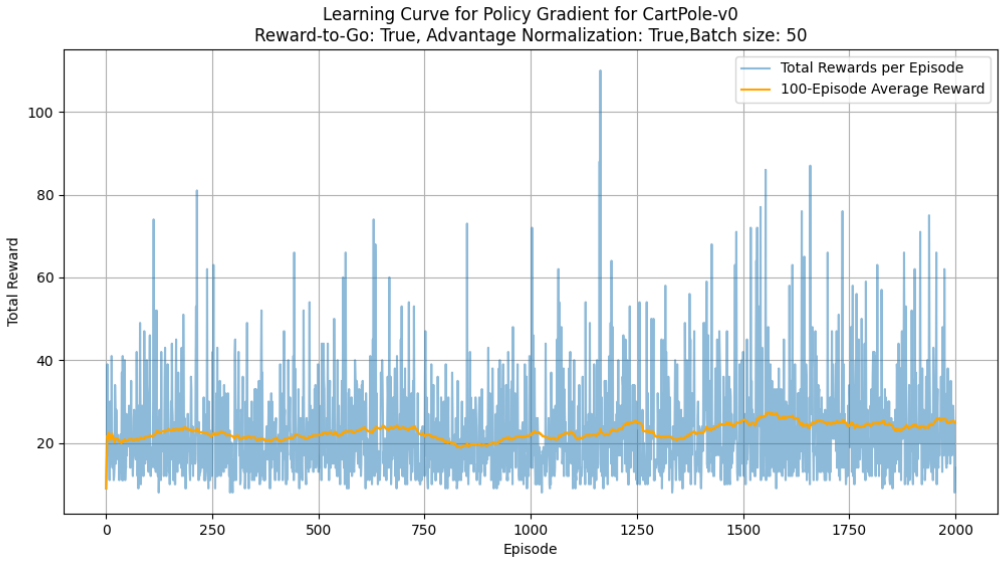
Q2(c)

Comparison of batch size
Generating batch size with 5,10,20,50


Learning Curve for Policy Gradient for CartPole-v0
Reward-to-Go: True, Advantage Normalization: True,Batch size: 5
Total Rewards per Episode
100-Episode Average Reward
Total Reward
Episode
0
25
50
75
100
125
150
175
200
0
250
500
750
1000
1250
1500
1750
2000


Learning Curve for Policy Gradient for CartPole-v0
Reward-to-Go: True, Advantage Normalization: True,Batch size: 10
Total Rewards per Episode
100-Episode Average Reward
Total Reward
Episode
0
25
50
75
100
125
150
175
0
250
500
750
1000
1250
1500
1750
2000





Observations

I experimented with batch size of 5,10,20 and 50
With smaller batch size the convergence is achieved with more time but it results in model learning better and fetching better rewards compared to increase in the batch size.