RL REPORT

Assignment-3

Vuppula Hemanth Reddy

AI23BTECH11033

Code:-Githublink

Problem-1

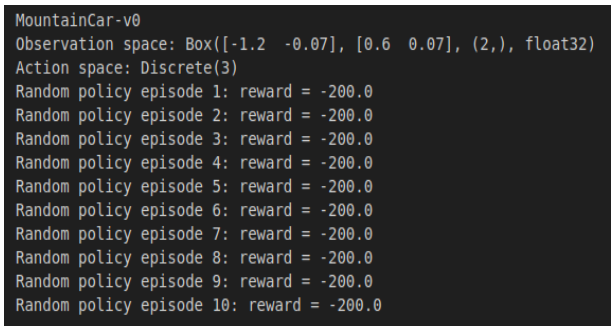
(a)

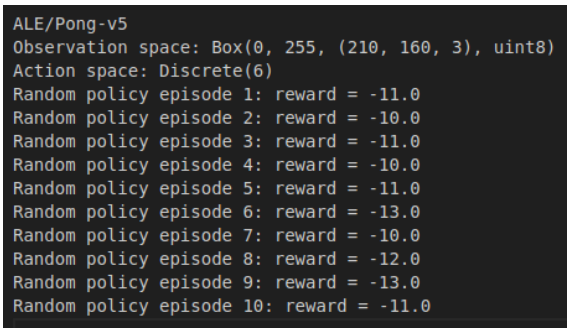
Code snippet:

```python
def probe_env(env_name, episodes=3, max_steps=500):
    print(f"{env_name}")
    env = gym.make(env_name)
    print("Observation space:", env.observation_space)
    print("Action space:", env.action_space)

    for ep in range(episodes):
        obs, info = env.reset()
        total_reward = 0
        for t in range(max_steps):
            action = env.action_space.sample()
            obs, reward, terminated, truncated, info = env.step(action)
            total_reward += reward
            done = terminated or truncated
            if done:
                break
        print(f"Random policy episode {ep+1}: reward = {total_reward}")
    env.close()
```

Output:-





Observations:-

1) Mountain car:
    - State space is box(x,) with continuous values with range of position as [–1.2,0.6] and velocity as [-0.07,0.07]
    - Discrete action space is (push left,do nothing,push right)
    - Reward function is -1 if goal is not reached
    - Max episode length is 200
2) Pong:
    - State space is Box(210, 160, 3) - RGB image frames
    - Discrete action space ={do nothing ,move up,move down,3 other parameters}
    - Reward function is +1 if agent scores a point and -1 if opponent scores a point 0 otherwise.Game is won if one reaches 21 game points.
    - Therefore episodic reward ranges from -21 to +21.
3) Random agent performs poorly on both the environments as the agent rarely reaches the goal in Mountain car and agent cannot track the ball trajectory.

(b)

Hyperparameters:-

- Batch size:no. of experiences sampled from replay buffer for each training step
- Gamma:discount factor
- Episodes:total number of training episodes the agent should runs
- Learning rate:learning rate for optimizer updating network weights during training
- Buffer capacity:Maximum no. of experiences stored in replay buffer for sampling
- Min buffer size:minimum experiences required in replay buffer before model

starts training
- Target update steps:no of steps between updates of target network
- Epsilon_start:initial value of epsilon
- Epsilon_end:least value of epsilon
- Epsilon decay:controls the rate at which epsilon decreases exponentially over time during training.
- Loss Function:MSE error
- Optimizer:Adam optimizer

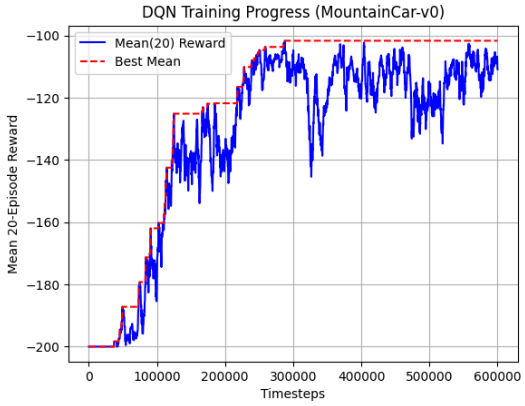
1) MountainCar-v0:

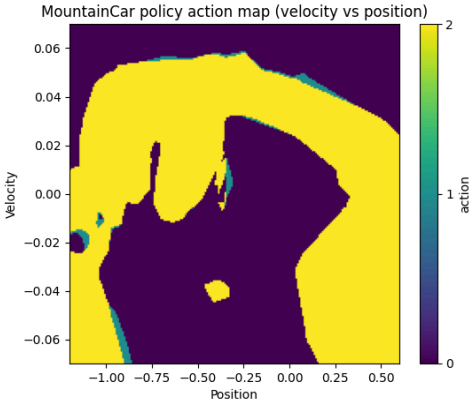
Hyperparameters used:

- Episodes:3000
- Batch size:64
- Gamma:0.99
- Learning rate:5e-4
- Buffer capacity:50000
- Min buffer size:1000
- Target update steps:500
- Epsilon start:1
- Epsilon end:0.05
- Epsilon decay:20000

After multiple trial and error ,I found this set of values supports efficient and stable DQN training in mountain car-v0.

plots:-





Observations:

- Training curve reaches a good mean reward of -110
- We can see that the agent learns via random exploration and reaches a good stable mean reward.
- The action map shows a meaningful decision boundaries and hence showcasing that the agent can build up the necessary momentum to reach the goal

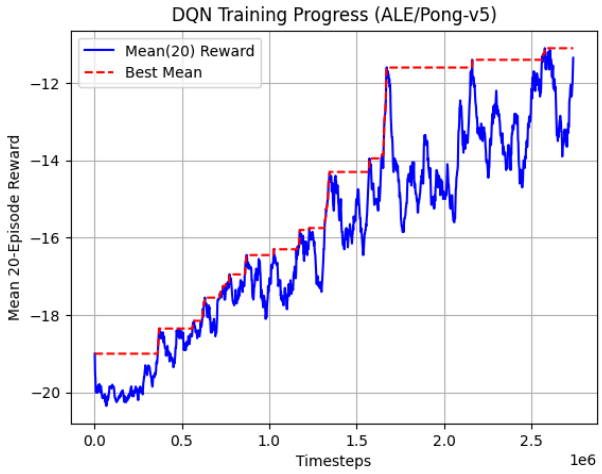
2) Pong:

Hyperparameters used:

- Episodes:1000
- Batch size:32
- Gamma:0.99
- Learning rate:2e-4
- Buffer capacity:10000
- Min buffer size:5000
- Target update steps:5000
- Epsilon start:1
- Epsilon end:0.05
- Epsilon decay:500000

I initially tried it with a bigger buffer size but the cpu crashed because of it hence after multiple crashes a buffer size of 10000 was helpful.Using these values of Hyperparameters it took around 2 days for 1000 episodes to be completed using a cpu.

Plots:-



Observations:-

- The agent reached a 20 episode mean reward of -11 in 2.5 million steps which shows that the model is improving.
- We can see that there is a stochastic nature in the training curve i.e it exhibits periodic ups and downs yet improves overall.
- Given more computational resources and time the agent could improve from -11 and can learn to get positive rewards.

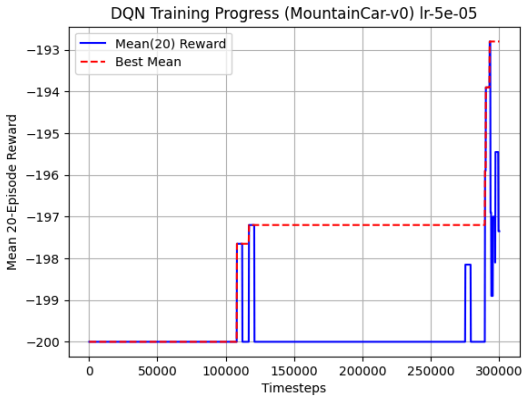
(c)

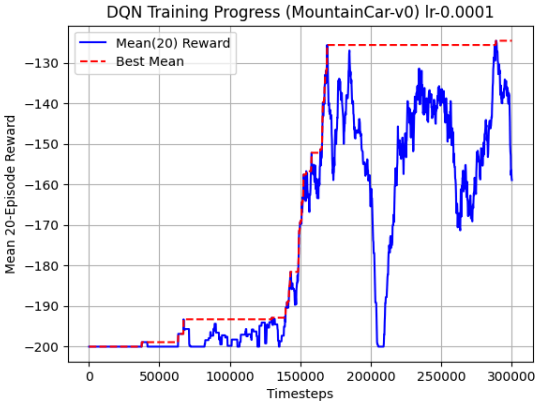
The hyperparameter chosen here is learning rate.

This hyperparameter tuning is done only on mountain cart and not on pong because of lack of computational resources.

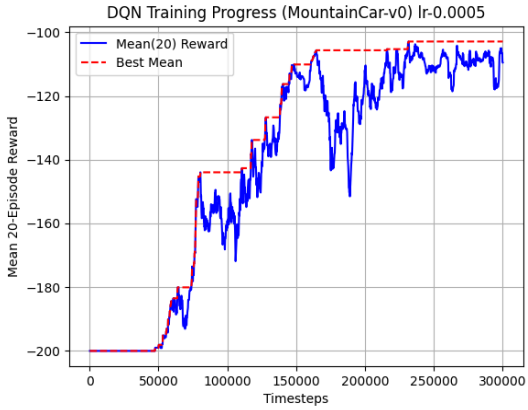
Plots:-

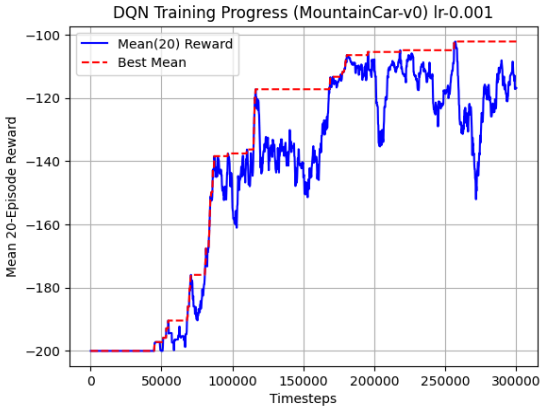
Learning curves:-



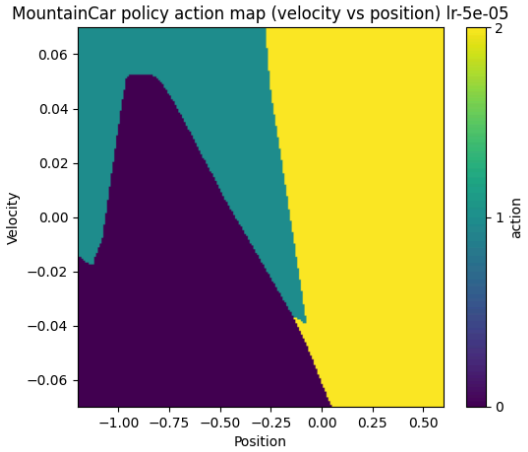


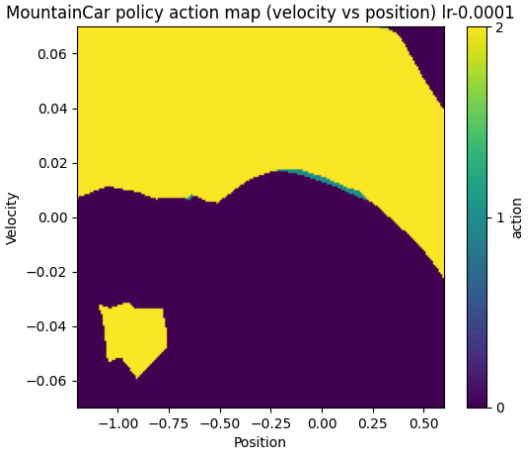
ac

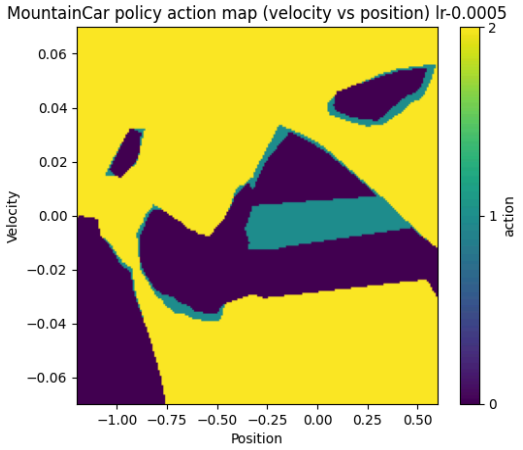


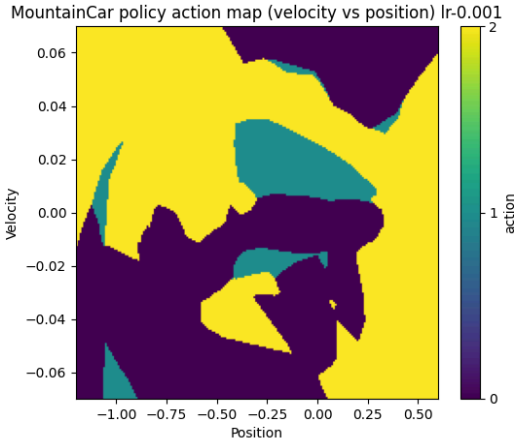


Policy action maps:-

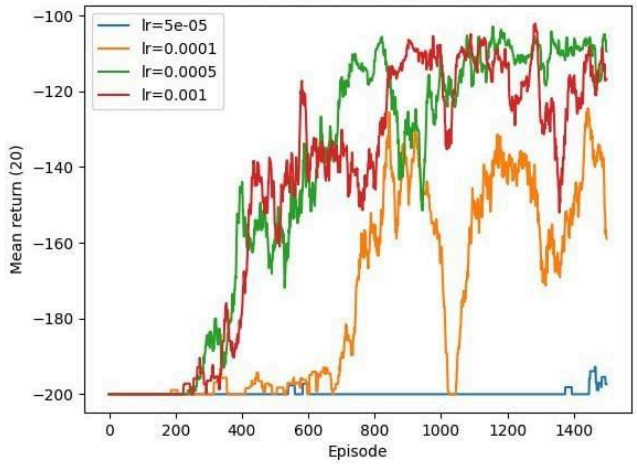








Final plot:-



Observations:-

1) LR 5e-5:learning is extremely slow and agent struggles to improve
2) LR 1e-4:there is an improvement compared to 1st LR but is still slow and mean rewards reaches less mean rewards compared to higher learning rate.
3) LR 5e-4:here learning is stable and is also faster .among all LR this is the most suitable learning rate for this environment.
4) LR 1e-3:although the agent learns quickly and achieves high rewards ,the performance is unstable at later episodes.i.e in later episodes the learning rate leads to rapid but unstable updates which we do not prefer.
5) At low LR the agent is stuck at near constant actions as shown in action map reflecting the agents inability to explore more effective strategies.
6) As learning rate increases we can see that the action maps display clear structured

regions reflecting effective learning. At LR1e-3 we can see that action map is non-uniform and is fragmented showing the instability we saw in learning curves.

7) Overall 5e-4 achieves best balance and hence a good choice for Mountain cart.

Problem-3

(a)

Code snippet:-

```python
def probe_env(env_name, episodes=3, max_steps=500):
    print(f"{env_name}")
    env = gym.make(env_name)
    print("Observation space:", env.observation_space)
    print("Action space:", env.action_space)

    for ep in range(episodes):
        obs, info = env.reset()
        total_reward = 0
        for t in range(max_steps):
            action = env.action_space.sample()
            obs, reward, terminated, truncated, info = env.step(action)
            total_reward += reward
            done = terminated or truncated
            if done:
                break
        print(f"Random policy episode {ep+1}: reward = {total_reward}")
    env.close()
```

Output:-

```
CartPole-v1
Observation space: Box([-4.8               -inf -0.41887903        -inf], [4.8               inf 0.41887903        inf], (4,), float32)
Action space: Discrete(2)
Random policy episode 1: reward = 13.0
Random policy episode 2: reward = 24.0
Random policy episode 3: reward = 20.0
Random policy episode 4: reward = 17.0
Random policy episode 5: reward = 38.0
Random policy episode 6: reward = 13.0
Random policy episode 7: reward = 23.0
Random policy episode 8: reward = 13.0
Random policy episode 9: reward = 10.0
Random policy episode 10: reward = 18.0
```

```
LunarLander-v3
Observation space: Box([ -2.5        -2.5       -10.        -10.         -6.2831855 -10.
  -0.         -0.       ], [ 2.5        2.5       10.        10.         6.2831855 10.
  1.         1.       ], (8,), float32)
Action space: Discrete(4)
Random policy episode 1: reward = -175.19409488179514
Random policy episode 2: reward = -2.000299155966715
Random policy episode 3: reward = -103.39159810185652
Random policy episode 4: reward = -260.98215248633926
Random policy episode 5: reward = -162.4680749718187
Random policy episode 6: reward = -306.9835866153568
Random policy episode 7: reward = -428.2427937542936
Random policy episode 8: reward = -224.64721249825377
Random policy episode 9: reward = -119.23994865870142
Random policy episode 10: reward = -101.1286529394615
```

Observations:-

1) CartPole:
    - Observation space is Box(4,0) namely (position, velocity, pole angle, pole angular velocity)
    - Action space is Discrete either move up left or right
    - Reward function is +1 per time step until episode ends
2) Lunarlander:
    - Observation space is Box(8,0)
    - Action space is Discrete (4)-do nothing,fire left engine ,fire right engine,fire main engine
    - Reward function depends on how the landing happens with penalties for unfavourable landing and bonus for landing successfully.
3) Random agent performs poorly on both.its highly variable in lunarlander and is less stable in cartpole as the agent cannot balance the pole for long.

(b)

Hyperparameters:

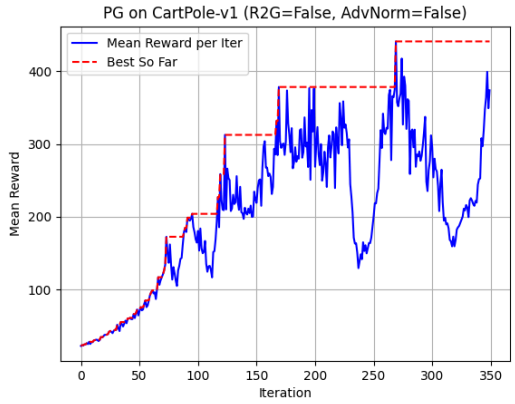
- iterations :No. of training loops to perform
- Batch size: total number of timesteps gathered before every policy update
- Learning rate
- Gamma:Discount factor
- Reward to go:if true,each step uses only the sum of rewards from that step onwards.if false,uses total return for all steps in an episode.
- Advantage norm:if false,advantage estimates are not normalized and if true it normalizes the computer advantages to zero mean unit variance.

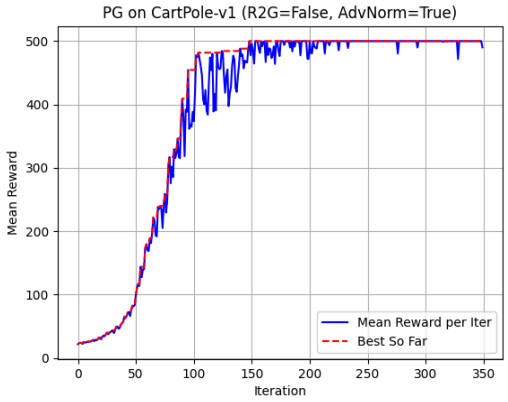
Cartpole:

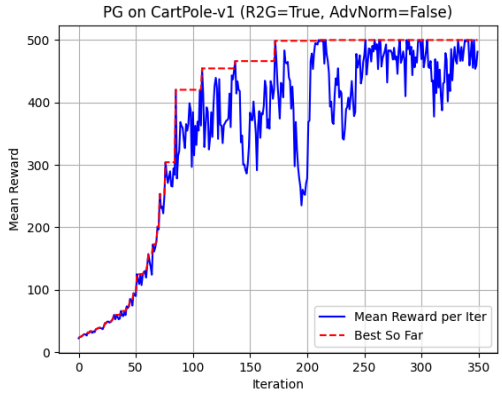
Hyperparameters used:

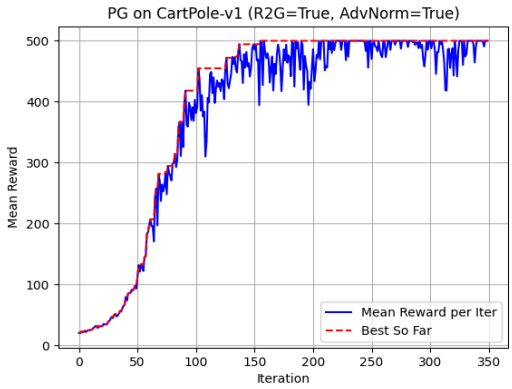
- Iterations:350
- Batch size:5000
- Learning rate:0.001
- gamma:0.99

Plots:-









Observations:-

- Using both R2G and AdvNorm leads to fastest and most stable learning.
- Enabling only R2G produces effective learning and reaches high rewards but exhibits large fluctuations at later iterations.
- Enabling only AdvNorm improves both speed and stability of learning i.e achieves

optimal performance quickly.

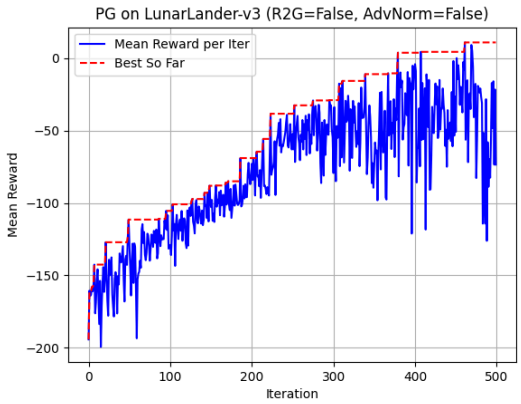
- Disabling both results in slow and less stable learning with mean rewards showing high variance.
- Hence, enabling both gives optimal performance.

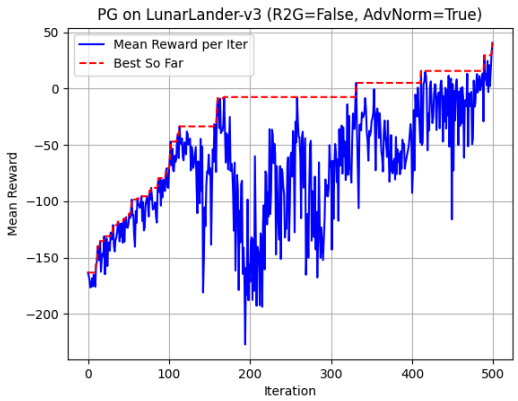
Lunarlander:

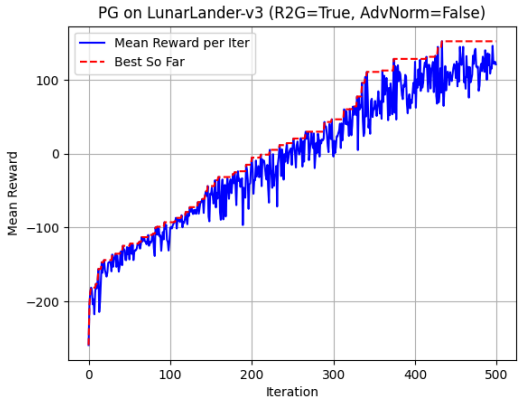
Hyperparameters used:

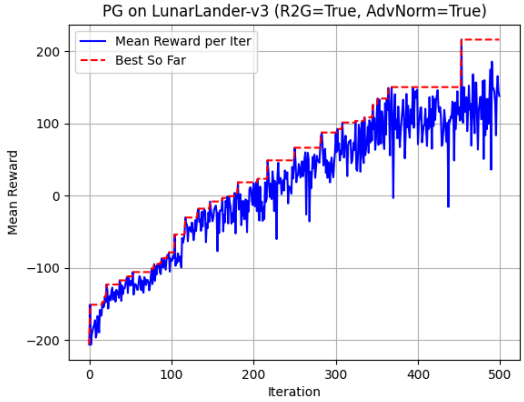
- Iterations:500
- Batch size:5000
- Learning rate:0.001
- gamma:0.99

Plots:









Observations:

- Enabling both yields fastest ,most stable reward improvement.
- Using only R2G achieves steady reward growth with less fluctuations compared to using only AdvNorm.
- Using only AdvNorm shows that mean rewards have high variance at later iterations but higher speed of convergence.
- Not using both results in slowest learning and high variance at later iterations.
- Hence using both gives the best learning for lunarlanding.

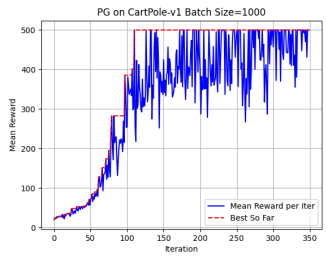
(c)

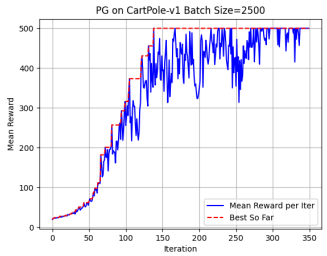
Cartpole:

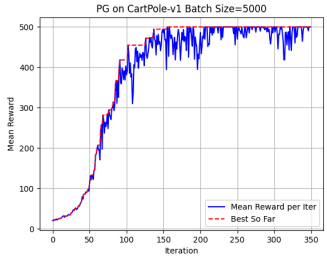
Hyperparameters used:

- Iterations:350
- Batch size:5000,2500,1000
- Learning rate:0.001
- gamma:0.99

Plots:-







Observations:

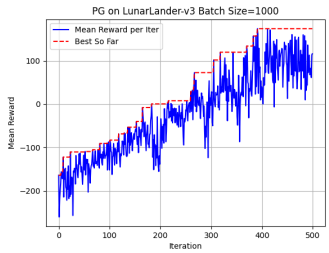
- All batch sizes eventually converge to max reward i.e 500.
- Larger batch sizes show smoother learning curves with less variance in mean rewards.
- Smaller batch sizes show more noisy gradient estimates resulting in more fluctuation in mean reward.
- Therefore larger bath size results in smoother and more stable learning in cartpole.

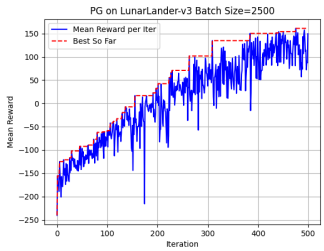
Lunarlander:

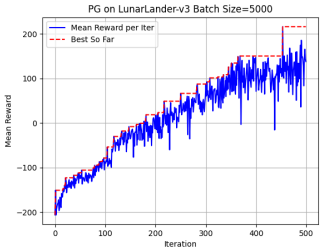
Hyperparameters used:

- Iterations:500
- Batch size:5000,2500,1000
- Learning rate:0.001
- gamma:0.99

Plots:-







Observations:-

- Same as before, larger batch sizes give smoother and stable mean rewards.
- But as batch sizes increase the agent achieves a larger mean reward at same iterations.
- Hence ,using higher batch size gives faster convergence,smoother and stable learning.