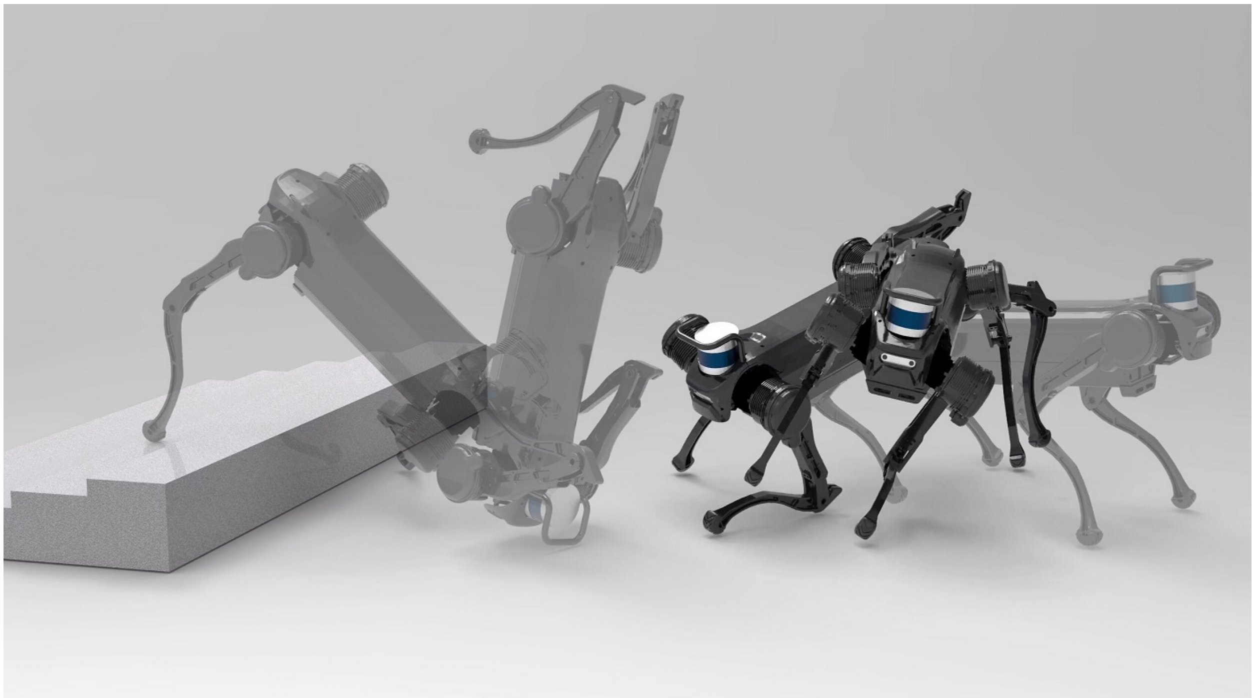
REPORT

Deep Learning for Robotics Applications Assignment 2



Suryaveer Singh

10/06/2023

OBJECTIVES

The primary objectives of this assignment are to:

1. Train and test OpenAI gym environments with stable baseline3 RL models.
2. Train and test one environment with Discrete Observation & Action Spaces and one environment with Continuous Observation Space using DQN.
3. Train and test two Continuous Environments with DDPG, PPO, or SAC.
4. Report findings and training process.

Part 1: Deterministic Policies

Environment 1: Cliff Walking

Training with Stable-Baseline DQN

In this section of the assignment, I chose an environment with discrete observation and action spaces and trained it using the Stable-Baseline implementation of DQN (Deep Q-Network).

Environment Details:

-This environment is part of the Toy Text environments. This is a simple implementation of the Gridworld Cliff reinforcement learning task.

- Observation Space: [Discrete] [48]

- Action Space: [Discrete][4]

Training Process:

1. **Model Initialization**: We initialized the DQN agent and the environment.

```
model = DQN('MlpPolicy',env,
            target_update_interval=target_update,
              exploration_fraction = exploration_fraction,exploration_final_eps=final_eps,
              learning_rate=lr,policy_kwargs=policy_kwargs,
              buffer_size=buffer_size,learning_starts=learning_starts,
              batch_size=batch_size,gamma=gamma,
              tensorboard_log='dqndiscrete/cliffwalking',
                verbose=1)
model.learn(total_timesteps=200000)
```

```

env = gym.make('CliffWalking-v0')

env = DummyVecEnv([lambda: env])

```

2. **Hyperparameters**: The following hyperparameters were used for training:

- Learning Rate: 0.0003
- Discount Factor (Gamma): [0.97]
- Epsilon (Exploration Rate): [0.15]
- Replay Buffer Size: [10_000]
- Batch Size: [128]
- Target Network Update Frequency: [100]

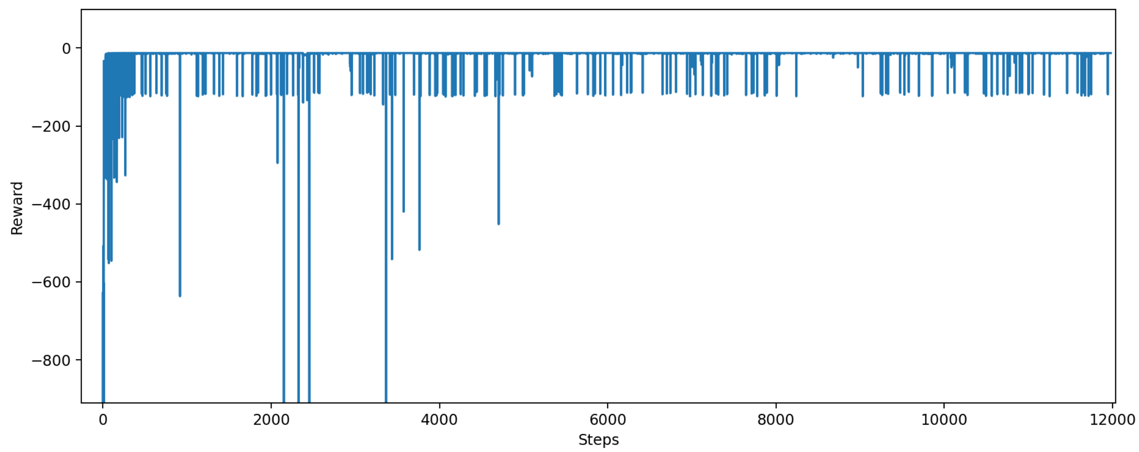
3. **Training**: The agent was trained for a total of 200000 episodes.

Can be run with file part1_discrete_cliffwalking_train.py

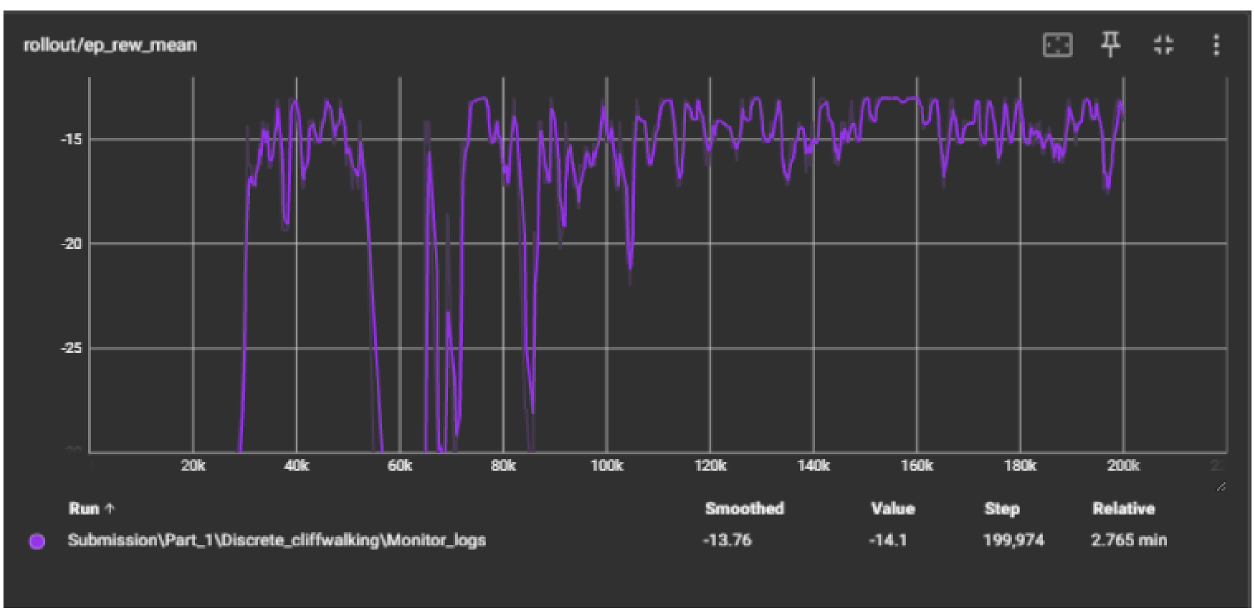
4. **Results**: The training process resulted in the following outcomes:

Can be run with file part1_discrete_cliffwalking_test.py

Reward Function History



Average Episodic Reward History



Environment 2: Lunar Lander (Discrete Action)

Training with Stable-Baseline DQN

In this section, we chose an environment with continuous observation space and trained it using the Stable-Baseline implementation of DQN.

Environment Details:

- I have chosen LunarLander-v2 which is part of the Box2D environments

- **Observation Space**: Continuous

Box([-1.5 -1.5 -5. -5. -3.1415927 -5. -0. -0.], [1.5 1.5 5. 5. 3.1415927 5. 1. 1.], (8,), float32)

- **Action Space**: [Discrete(4)]

Training Process:

Can be run with file part1_continuos_lunar_lander_train.py

1. Model Initialization: I initialized the DQN agent and the environment with gym and stablebaseline3.

```
# Create the LunarLander-v2 environment
env = gym.make('LunarLander-v2')

env = Monitor(env,'dqn_continuos_lunar_lander')

policy_args = dict(net_arch = [256,256])
# Define and create the DQN agent
model = DQN('MlpPolicy',learning_rate=0.00063,batch_size=128,
            buffer_size=50000,exploration_final_eps=0.1,exploration_fraction=0.12,
            gamma=0.99,gradient_steps=-1,learning_starts=0, target_update_interval=250,
            train_freq=4,policy_kwargs=policy_args,env=env,tensorboard_log="dqn_continuos/"

# Train the agent
```

2. Hyperparameters: The hyperparameters employed for training were:

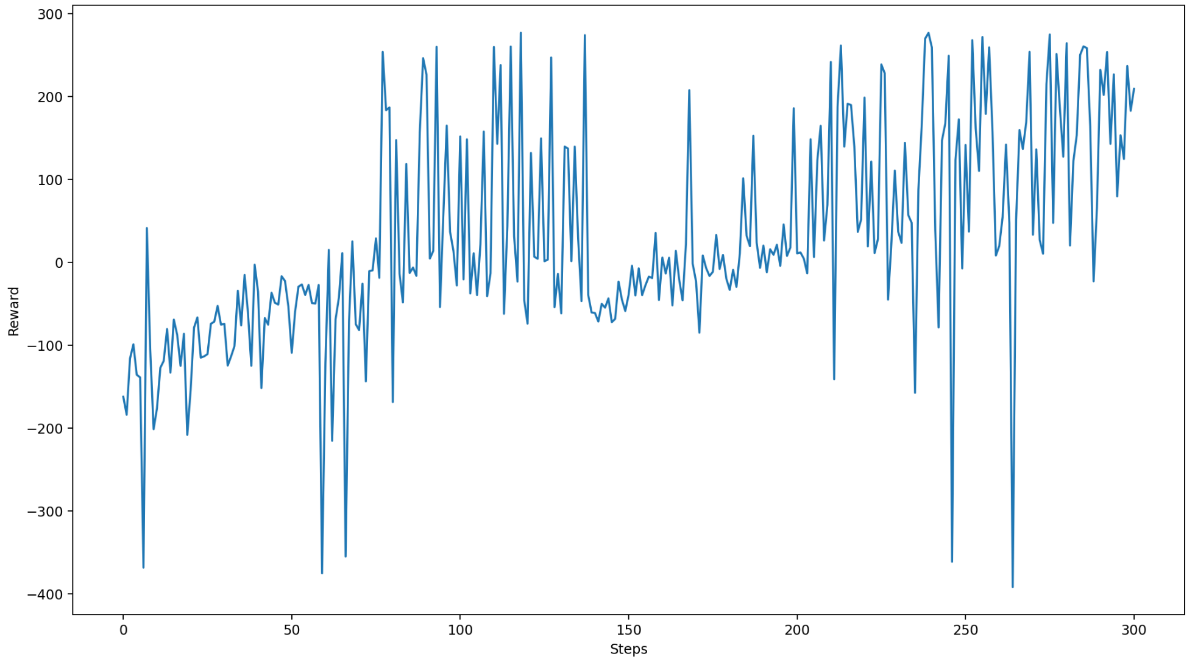
- Learning Rate: 0.00063
- Discount Factor (Gamma): [0.99]
- Epsilon (Exploration Rate): [0.12]
- Replay Buffer Size: [50_000]
- Batch Size: [128]
- Target Network Update Frequency: [250]
- Model DQN arch : net_arch = [256,256]

3. **Training**: The agent was trained for a total of 100000 episodes.

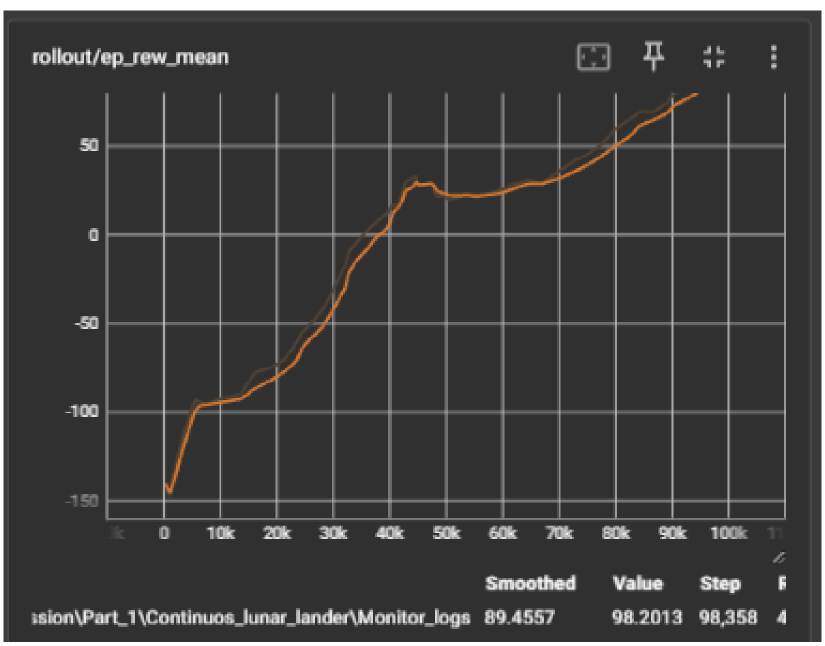
4. **Results**:The training process resulted in the following outcomes:

Can be run with file Part1_continuos_lunar_lander_test .py

- Reward Function History:



- **Average Episodic Reward History:**



Part 2: Policy Gradient

Environment 3: Pendulum-v1

Training with Stable-Baseline **PPO**

In this part of the assignment, we selected a continuous environment and trained it using the Stable-Baseline implementation of PPO(Proximal Policy Optimization).

Environment Details:

- **Pendulum-v1** from the Classic Control environments.

- Observation Space: [Continuous Box([-1. -1. -8.], [1. 1. 8.], (3,), float32)]

- Action Space: [Continuous Box(-2.0, 2.0, (1,), float32)]

Training Process:

1. **Initialization**: We initialized the PPO agent and the environment.

```
env = gym.make('Pendulum-v1')

env = DummyVecEnv([lambda: env])
```

```
env = Monitor(env,'ppo_pendulam')


model = PPO(verbose=1,tensorboard_log='ppocontinuos/pendulum',policy=policy,env=env,
            clip_range=clip_range,
            ent_coef=ent_coef,gae_lambda=gae_lambda,
            gamma=gamma,learning_rate=learning_rate,
            n_steps=n_steps,n_epochs=n_epochs,
            sde_sample_freq=sde_sample_freq,use_sde=use_sde)


model.learn(total_timesteps=n_timesteps)
```

2. **Hyperparameters**: The hyperparameters specific to PPO were used for training:

```
learning_rate = 0.001
clip_range =  0.2
ent_coef = 0.0
gae_lambda = 0.95
gamma = 0.9
n_epochs = 10
n_steps = 1024
n_timesteps = 1000000
policy='MlpPolicy'
sde_sample_freq = 4
use_sde = True
```

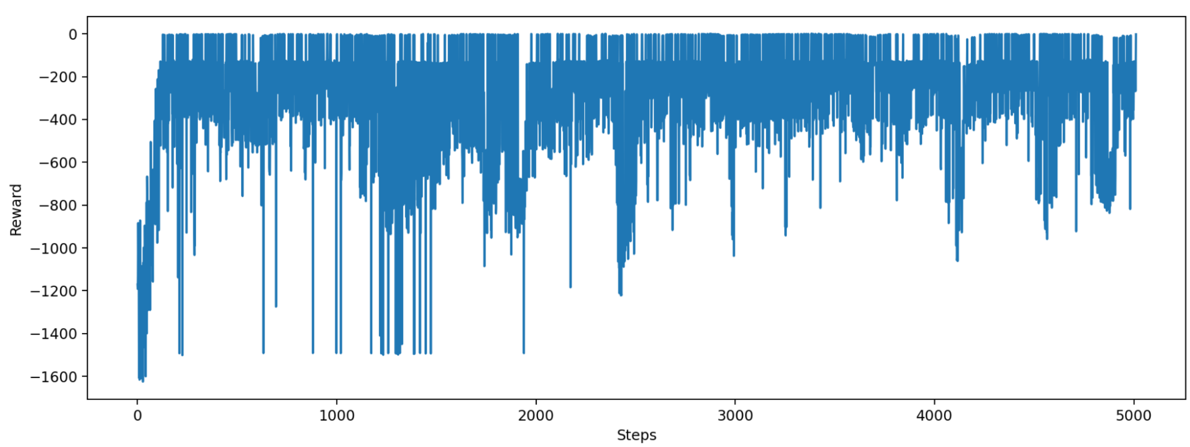
3. **Training**: The agent was trained for a total of 1000000 episodes.

Can be run with file part2_continuos_pendulam_train.py

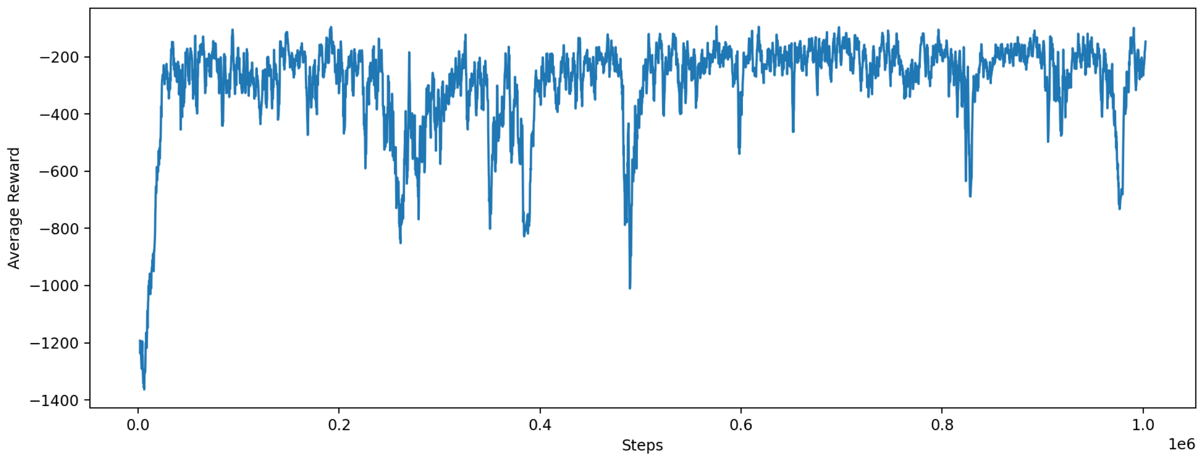
4. **Results**: The training process resulted in the following outcomes:

Can be run with file part2_continuos_pendulam_test.py

- Reward Function History:



- Average Episodic Reward History:



Environment 4: Car Racing

Training with Stable-Baseline PPO

In this part, I selected the second continuous environment of Car Racing and trained it using the Stable-Baseline implementation of PPO (Proximal Policy Optimization).

Environment Details:

The Car Racing environment is part of the Box2D environments. The goal is to drive over all the tiles in a lap. The reward is -0.1 every frame and +1000/N for every track tile visited, where N is the total number of tiles visited in the track. The episode finishes when all the tiles are visited. The car can also go outside the playfield - that is, far off the track, in which case it will receive -100 reward and die.

- Observation Space: [Box(0, 255, (96, 96, 3), uint8)]

- Action Space: [Box([-1. 0. 0.], 1.0, (3,), float32)]

Training Process:

1. **Initialization**: I initialized the PPO agent and the environment.

```
def make_env(env_id, rank, seed=0):
    """
    Utility function for multiprocessed env.

    :param env_id: (str) the environment ID
    :param seed: (int) the inital seed for RNG
    :param rank: (int) index of the subprocess
    """

    def _init():
        env = gym.make(env_id)
        # use a seed for reproducibility
        # Important: use a different seed for each environment
        # otherwise they would generate the same experiences
        env.reset(seed=seed + rank)
        return env

    set_random_seed(seed)
    return _init

n_envs= 8  # You can adjust the number of parallel environments
env = SubprocVecEnv([make_env('CarRacing-v2', i + 32) for i in range(n_envs)],start_method='fork')
```

```
model = PPO('CnnPolicy', env, policy_kwargs=policy_kwargs,
            verbose=1,tensorboard_log='ppocontinuos/',
            batch_size=batch_size, n_steps=n_steps, gamma=gamma, gae_lambda=gae_lambda,
            ent_coef=ent_coef, vf_coef=vf_coef, max_grad_norm=max_grad_norm,
            learning_rate=learning_rate, use_sde=use_sde, clip_range=clip_range,
            sde_sample_freq=sde_sample_freq, n_epochs=n_epochs,
            )
```

2. **Hyperparameters**: The hyperparameters specific to PPO were used for training:

```
n_timesteps= 4e6
policy= 'CnnPolicy'
batch_size= 128
n_steps= 512
gamma= 0.99
gae_lambda= 0.95
n_epochs= 10
ent_coef= 0.0
sde_sample_freq= 4
max_grad_norm= 0.5
vf_coef= 0.5
learning_rate= 1e-4
use_sde= True
clip_range= 0.2
policy_kwargs= dict(log_std_init=-2,ortho_init=False, activation_fn=nn.GELU, net_arch=dict(pi=[256], vf=[256]))
```

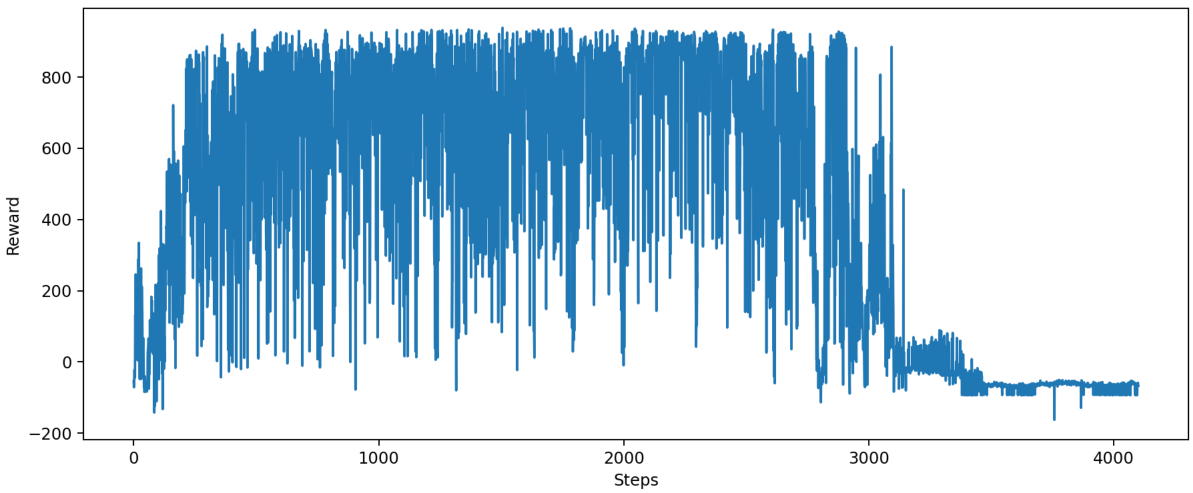
3. **Training**: The agent was trained for a total of 4000000 episodes.

Can be run with file part2_continuos_raceing_train.py

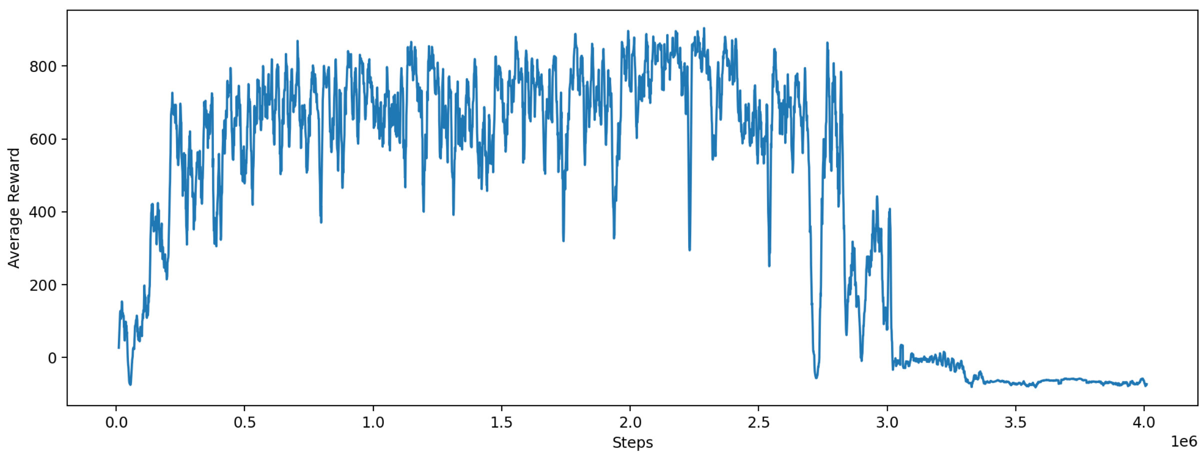
4. **Results**: The training process resulted in the following outcomes:

Can be run with file part2_continuos_raceing_test.py

- Reward Function History:



- Average Episodic Reward History:



Bonus: MuJoCo Walker2D

For the bonus part of the assignment, I chose the MuJoCo Walker2D environment and trained it with the stable baselines3 implementation of PPO algorithm. The goal was to achieve successful training, enabling the walker to consistently walk forward.

Action Space: Box(-1.0, 1.0, (6,), float32)

Observation Space: Box(-inf, inf, (17,), float64)

The reward consists of three parts:

healthy_reward: Every timestep that the walker is alive, it receives a fixed reward of value healthy_reward,

forward_reward: A reward of walking forward which is measured as forward_reward_weight * (x-coordinate before action - x-coordinate after action)/dt. dt is the time between actions and is dependent on the frame_skip parameter (default is 4), where the frametime is 0.002 - making the default dt = 4 * 0.002 = 0.008. This reward would be positive if the walker walks forward (positive x direction).

ctrl_cost: A cost for penalizing the walker if it takes actions that are too large. It is measured as ctrl_cost_weight * sum(action2) where ctrl_cost_weight is a parameter set for the control and has a default value of 0.001

The **total reward** returned is reward = healthy_reward bonus + forward_reward - ctrl_cost and info will also contain the individual reward terms

Training Process:

1. **Initialization**: I initialized the agent and the MuJoCo Walker2D environment.

```
# env = gym.make("Walker2d-v4")
vec_env = DummyVecEnv([lambda: gym.make("Walker2d-v4")])

# vec_env = Monitor(vec_env,'walker2d_mujoco')

# env = gym.wrappers.RecordEpisosdeStatistics(env)
vec_env = VecNormalize(vec_env, norm_obs=True, norm_reward=True,
                        clip_obs=10.)
```

```
# Train the agent

model = PPO("MlpPolicy", vec_env,batch_size=batch_size,
            n_steps=n_steps,
            gamma=gamma,learning_rate=learning_rate,
            ent_coef=ent_coef,clip_range=clip_range,
            n_epochs=n_epochs,gae_lambda=gae_lambda
            ,max_grad_norm=max_grad_norm,vf_coef=vf_coef,
                verbose=1, tensorboard_log="a2cWalker2d")

model.learn(total_timesteps=n_timesteps,progress_bar=True)
```

2. **Hyperparameters**: The following hyperparameters were used for training:

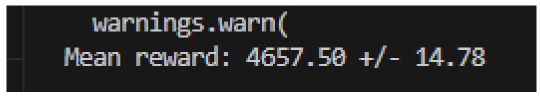
```
n_timesteps= 1e6
batch_size= 32
n_steps= 512
gamma= 0.99
learning_rate= 5.05041e-05
ent_coef= 0.000585045
clip_range= 0.1
n_epochs= 20
gae_lambda= 0.95
max_grad_norm= 1
vf_coef= 0.871923
```

3. **Training**: The agent was trained for 1000000 episodes to achieve consistent forward walking. **Can be run with file part3_walker_2d_train.py**

4. **Results**: The bonus task resulted in the following outcomes:

Can be run with file part3_walker_2d_test.py

- Achievement of consistent forward walking.

Mean Rewards while testing : 

References:

Gymnasium: https://gymnasium.farama.org/environments/classic_control/

SB3: https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html
Rl-zoo: https://stable-baselines3.readthedocs.io/en/master/guide/rl_zoo.html