RL project

Aayush Kucheria (781798), Adrian Müller (918804)

ELEC-E8125 - Reinforcement Learning

December 14, 2022

Part 1

Task 1

We trained Policy Gradient (PG) and Deep Deterministic Policy Gradient (DDPG) with 3 random seed, both on LunarLander-v2 (medium level) and BipedalWalker-v3 (easy level). For both environments, the required target, 100 for LunarLander-v2 (medium) and 250 for BipedalWalker-v3 (easy), respectively, was achieved using DDPG. In Table 2, the average rewards of 50 test episodes for each algorithm, environment, and seed are shown and additionally averaged over all three seeds.

seed	870	601	1	**Avg.**
DDPG on LL	206.8 (111.9)	215.3 (83.2)	176.7 (139.3)	199.6 (115.0)
DDPG on BW	291.2 (0.7)	269.6 (79.6)	288.2 (41.3)	283.0 (52.6)
PG on LL	94.5 (130.2)	186.2 (98.3)	211.4 (88.6)	164.1 (118.4)
PG on BW	-125.0 (8.4)	-100.8 (0.4)	-119.6 (0.0)	-115.1 (11.5)

Table 1: Summary of the test runs (50 episodes) of all trained models. Numbers show the average over test rewards, with the standard deviation in parentheses. All figures are rounded to one decimal place. LL = LunarLander-v2 (medium), BW = BipedalWalker-v3 (easy).

The figures below show the comparative performances of DDPG and PG in both environments, each combination averaged over all three training runs (seeds 870, 601, and 1), i.e., in total 12 training runs are shown. If possible, all algorithms were trained until convergence. In Figure 1, neither DDPG nor PG showed further positive change in reward in subsequent episodes, such that we stopped training after 2000 and 3500 episodes, respectively. In Figure 2, PG was trained up to episode 3000, but still no learning occurred, so the graph was trimmed to 1300. DDPG for BipedalWalker-v3 (easy) showed a decrease in reward in subsequent training episodes, such that we stopped training after 1300 episodes.

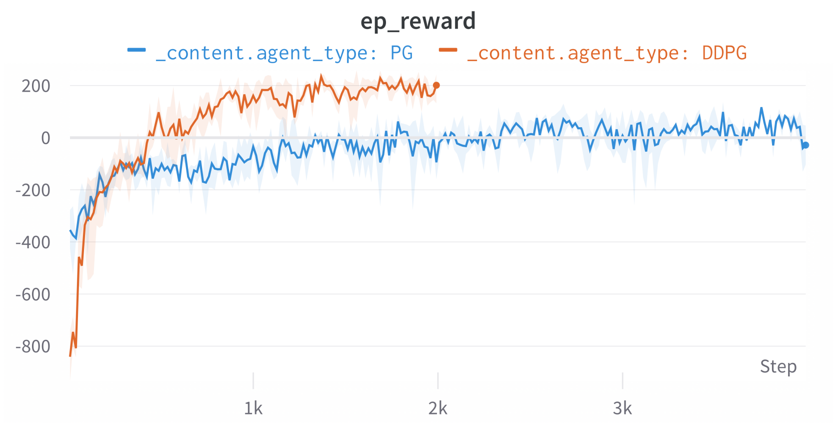

Figure 1: Training performance of DDPG and PG trained on LunarLander-v2 (medium)

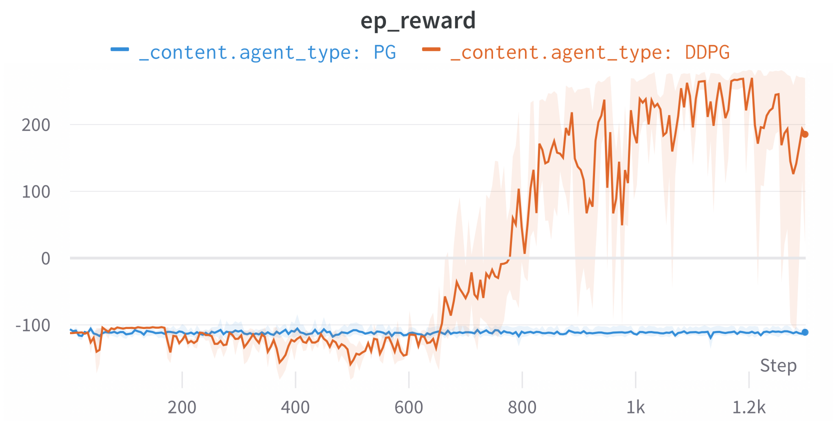

Figure 2: Training performance of DDPG and PG trained on BipedalWalker-v3 (easy)

Question 1

In both environments, DDPG performed better. Below is the average clock-time for each algorithm-environment pair.

	runtime
DDPG on LL	1h 20min
DDPG on BW	3h 50min
PG on LL	1h 55min
PG on BW	40min

Table 2: Average runtime of the training phase for all algorithm-environment pairs.

DDPG does better than PG on the investiged OpenAI gym environments because it is an actor-critic algorithm, i.e., it also relies on a critic to provide a value function that guides the policy. DDPG is better suited than PG for continuous action spaces. Additionally, DDPG algorithms can use an experience replay memory which stores and samples past experiences in order to more easily recognize patterns and generalize better than PG, which makes it better suited to more complex OpenAI gym environments, like the BipedalWalker environment.

Even though PG was trained way longer than DDPG on LunarLander (4000 vs. 2000 episodes), it still didn't reach the same average rewards. Even stopping at the same real time (1h 20min), the average reward isn't as high as with DDPG. Therefore, DDPG proved more sample efficient. For the BipedalWalker environment, PG seemed not to have learned anything. We tried different parameters, but the environment seemed to complicated for it to learn anything. Therefore, also here DDPG was more sample efficient.

Question 2

For DDPG, the most critical change was to use normal action noise, instead of uniform action noise. This allows to train "correct" actions, i.e., the mean of the noise distribution, instead of the lower end of it, and also explores sometimes more extreme actions. Also, we used more complex neural networks: For LunarLander, all networks (both in DDPG and the policy in PG) had 3 layers with unit counts 512, 256, and 64, and similarly, networks had 3 layers with 512, 512, and 64 units for BipedalWalker. For the BipedalWalker, we used also used Xavier initialization for the layers in both neural networks. We tried different learning rates, but they didn't seem to affect the training much.

Interestingly, the DDPG seemed to *unlearn* good behavior again. Figure 3 below shows the q-values of the critic in DDPG (seed: 1), trained until episode 4000. You can clearly see how the graph peaks at about episode 1300 and steadily declines in subsequent episodes. Hence, we cut off the training after episode 1300 in the runs documented above. Similarly, in the LunarLander environment, the training was cut after 2000 training episodes.

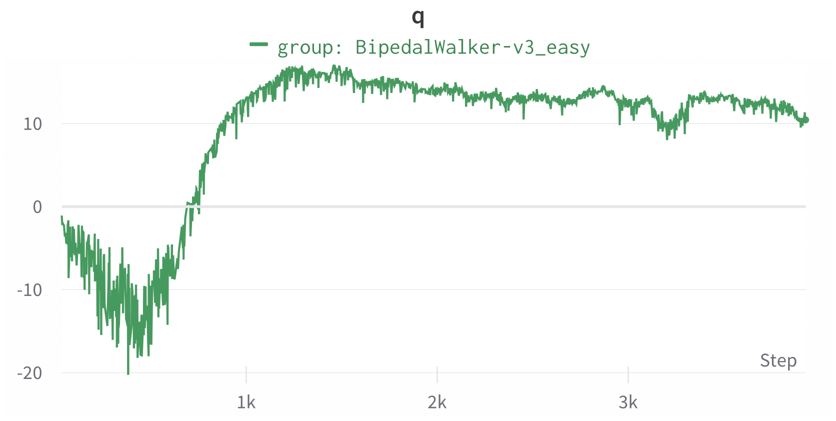

Figure 3: Q-values of DDPG trained on BipedalWalker-v3 (easy) until episode 4000

Part 2

Question 1

The research paper we picked was Proximal Policy Optimization Algorithms.

It presents a new reinforcement learning algorithm for training policies, called Proximal Policy Optimization (PPO). The main idea behind PPO is to directly optimize the policy by using a combination of several strategies, including trust region optimization and sample-based estimates of the policy gradient.

One of the key insights behind PPO is that policy gradient methods, which are commonly used in reinforcement learning, can be sensitive to the scale of the actions taken by the policy. This can cause the algorithm to make inefficient updates, which can slow down the training process. PPO addresses this issue by using a technique called "clipping," which scales down the policy gradient if it falls outside of a certain range, thus making the updates more efficient and stable.

Another important idea behind PPO is the use of a "value function" to predict the future reward that will be received by the agent. This value function is used to improve the performance of the algorithm by helping it to select actions that are likely to maximize the future reward. PPO uses a variation of the classic "actor-critic" algorithm, in which the value function is used to "criticize" the actions taken by the policy, and the policy is updated to better match the predictions of the value function.

Overall, the main ideas behind PPO are to improve the efficiency and stability of policy gradient methods in reinforcement learning by using techniques such as clipping and value function estimation. These ideas are intended to make PPO a more effective and reliable algorithm for training policies in a variety of environments.

Task 1

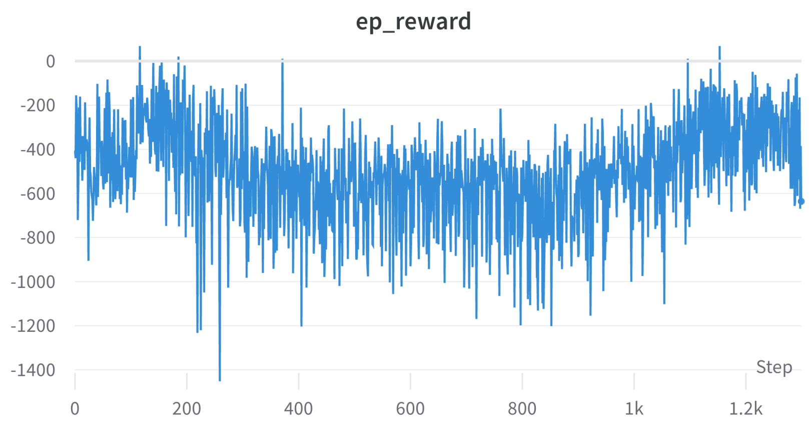

Figure 4: Training performance for PPO trained on LunarLander-v2 continuous (medium)

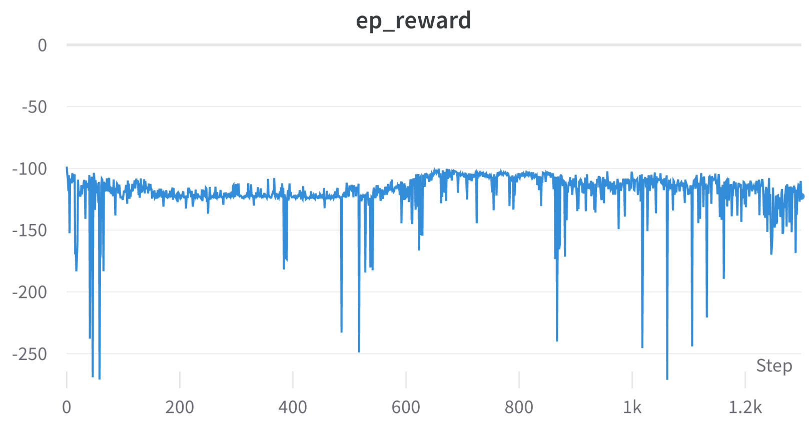

Figure 5: Training performance for PPO trained on BipedalWalker-v3 (easy)

Question 2

No it didn't. This is mainly because even though we have a running PPO algorithm built on top of our baseline algorithms, it still has some logical bugs and is in need of some hyperparameter tuning that we were unable to work on due to time constraints.

Task 2

Question 3

I expect the upgraded algorithms to cope with increased environment complexity.

One of the key strengths of PPO is its ability to handle large and continuous action spaces, which can be a challenging problem for many reinforcement learning algorithms. This could make PPO well-suited to environments where the number of possible actions is high, or where the actions can take on a wide range of values. Additionally, the use of a value function in PPO can help the algorithm to make more informed decisions and to better explore the environment, which could improve its performance in complex environments.

However, it is important to note that PPO, like any reinforcement learning algorithm, is not a silver bullet and will not be able to handle all types of environments. As the complexity of the environment increases, it may become necessary to use more sophisticated methods or to incorporate additional techniques and insights to effectively train a policy. In general, the ability of PPO to cope with increased complexity will depend on the specific details of the problem at hand.

Conclusion

DDPG works nicely on LunarLander-v2 (medium), as well as BipedalWalker-v3 (easy). PG did only work well on the LunarLander environment, but not in the BipedalWalker environment, and PPO seemed not to work well for either environments.