Deep Reinforcmenet Learning assignment

Laura Gozzo

June 2023

1 Hopper-v4

I chose to use PPO to train on the Hopper-v4 environment mainly for 3 reasons: 1- It is a state-of-the-art DRL algorithm (known to achieve great performance on many DRL tasks such as playing ATARI, or controlling a robotic arm), 2- It does not necessarily need hyperparameter tuning, and 3- it is computationally efficient. PPO is known for the following 3 advantages: simplicity, stability, and sample efficiency [3]. Simplicity means that compared to similar algorithms such as TRPO, PPO approximates what TRPO does and requires less computation. Stability means that its performance is generally stable across different environments. Sample efficiency means that, compared to other on-policy methods, it requires less data to achieve a good performance. This is because it uses each collected sample of data multiple times through epochs of minibatch updates. PPO updates the policy using a clip function, which ensures that policy updates are not too large and keeps the learning stable (and makes it more stable and more likely to converge to an optimal solution). These factors ensure good performance and a simple implementation, without the need for extensive fine-tuning. To implement PPO, the only change to the hyperparameters I made was adding an entropy coefficient of 0.01 to balance exploration and exploitation. This makes the algorithm slightly more likely to try out random actions while still mostly exploiting the most rewarding action known. If exploration is too low, the algorithm is more likely to converge too early to a suboptimal policy.

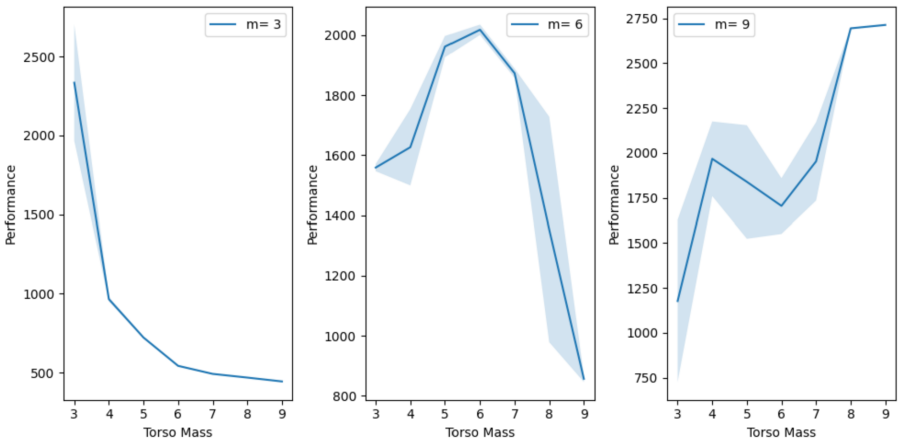

Figure 1: PPO performance of models trained on masses 3,6,9 across different masses

2 BipedalWalker-v3

To train on the environment BipedalWalker-v, I chose to use PPO as the on-policy algorithm and TQC as the off-policy algorithm.

I chose PPO for the advantages illustrated above, namely its simplicity and computational efficiency. These advantages prove to be very useful now that the hyperparameters have to be finetuned to reach a certain performance. Having fewer hyperparameters to tune, and the model taking a short time to run, allows for more efficient exploration of the hyperparameters. Since the benchmark score for PPO on Hugginface was much lower than 300 (288) [1], I did not use the same hyperparameters, instead, I used OPTUNA to optimize the learning rate and entropy coefficient. After 10 trials with 1.000.000 timesteps, the learning parameter and the entropy were optimized and the model achieved a score of 294. After finding the optimal parameters, PPO was re-run with 2.000.000 steps to achieve a better performance since it is an algorithm that can take many time steps to converge, it then reached a score of 296.

I chose to use TQC because it is known to outperform state-of-the-art algorithms on all environments from the continuous control benchmark suite [4]. TQC combines SAC, TD3, and QR-DQN, and uses quantile regression to predict a distribution for the value function instead of a mean value. The advantage of TQC is that it controls the overestimation bias (meaning that the values of actions tend to be overestimated), which is one of the main obstacles to off-policy learning. Because TQC is a very computationally expensive method, to fine-tune its hyperparameters I used the values indicated by the Huggingface benchmark [2], which should already be optimal. This achieved a mean reward of 335.68 at evaluation. Since it already achieved a very high reward and training took 2+ hours (with 1 million time steps and using Google Colab Pro), there

was no need to change the hyperparameters further. Because this model took a very long time to run, to investigate whether it could still achieve a good score with the least amount of time steps possible I implemented early stopping which stopped the training if it reached a reward of 300. This significantly decreased the running time to only 26 minutes and still performed well with a mean reward of 311 and only 67317 timesteps. Even with only 67317 timesteps, the reward and loss plots from tensorboard showed stable learning (Figures 2,3,4). Both the actor and critic loss plots flatten at a value close to 0. While the reward value doesn't seem to flatten, meaning that it could have increased further, it is still an impressive performance given the number of time steps.

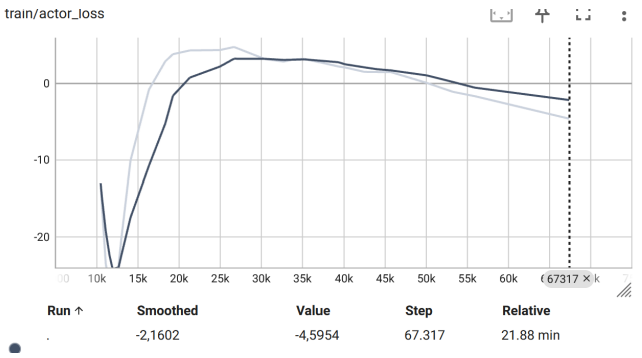

Figure 2: Actor Loss with Early Stopping

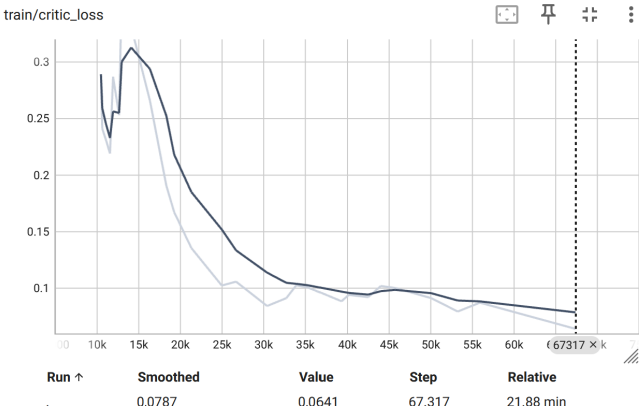

Figure 3: Critic Loss with Early Stopping

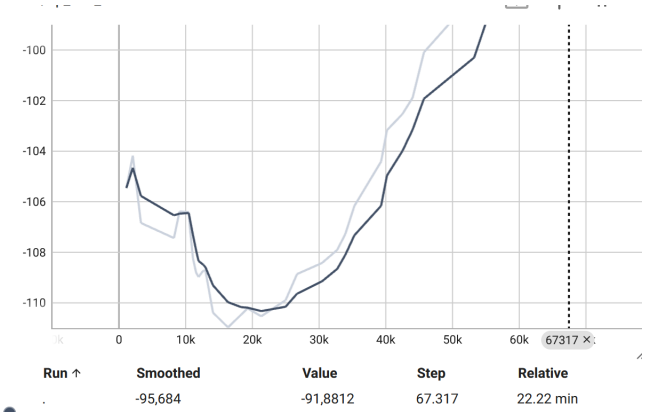

Figure 4: Mean Reward with Early Stopping

Compared to TQC, PPO was much more simple to implement (less hyperparameters to define) and simpler to understand. It is much less complex and requires less computational power to run each timestep as it involves relatively simple operations (computing gradients and performing policy updates) compared to TQC (which involves more complex calculations such as quantile regression). TQC has the disadvantage of being very computationally intensive and taking a long time to run each timestep.

Although PPO is sample-efficient for being an on-policy method, is it still much less sample-efficient than TQC, and needed more interactions with the environment to converge. TQC required more powerful hardware, however, PPO took many more timesteps to converge to an optimal policy (2.000.000 vs 67317), making it much slower than TQC when early stopping is implemented, and making TQC much more efficient.

PPO required more finetuning to achieve a decent performance compared to TQC.

In terms of maximum score achieved, TQC performed better than PPO. PPO can achieve relatively high scores for an on-policy method, however, because it makes small updates to the policy, it can also get stuck in a local minimum and not achieve the maximum score. TQC reaches much higher scores in different environments much more consistently. It uses probability distribution over the value function, which provides a finer understanding and better learning.

Furthermore, the performance of PPO varies a lot more with different seeds compared to TQC which has a more stable performance and is more robust to noise, making it more predictable and reproducible.

When comparing PPO and TQC on such tasks, it is important to consider that TCQ is made for continuous tasks. PPO has the advantage of being applicable in both discrete and continuous tasks, making it more versatile overall.

References

[1] Hugging Face. Ppo-bipedalwalker-v3. https://huggingface.co/sb3/ppo-BipedalWalker-v3. Accessed: 2024-06-23.

[2] Hugging Face. Tqc-bipedalwalker-v3. https://huggingface.co/sb3/tqc-BipedalWalker-v3. Accessed: 2024-06-23.

[3] Hugging Face. Everything you need to know about proximal policy optimization. https://huggingface.co/blog/deep-rl-ppo, 2023. Accessed: June 23, 2024.

[4] Arsenii Kuznetsov, Pavel Shvechikov, Alexander Grishin, and Dmitry Vetrov. Controlling overestimation bias with truncated mixture of continuous distributional quantile critics. In *International Conference on Machine Learning*, pages 5556–5566. PMLR, 2020.