Aidan Beesley
CIS 730: Principles of Artificial Intelligence

Introduction

For this project, I worked on hyperparameter tuning two algorithms in the LunarLander environment from OpenAI Gym, specifically Proximal Policy Optimization (PPO) and Advantage Actor Critic (A2C). After both algorithms were trained, they were compared to a pretraine baseline PPO model, with success measured primarily by the reward function of these algorithms, with the time taken to complete these algorithms taken as a secondary metric of success.

Background and Related Work

I have no background in AI, Reinforcement Learning, or using OpenAI Gym at all, this entire project was a learning experience for me.

Both PPO and A2C are policy optimization, on-policy, model-free algorithms (OpenAI). Policy optimization means that both attempt to optimize the parameters of a policy so that the policy will reach its desired goal state in the long term. On-policy means that the algorithms can only use the data from their most recent runs to learn from. Model-free means that the algorithms are not allowed to look ahead in their environment to plan, as say a chess bot would. The difference between the two algorithms is in how they attempt to maximize performance. A2C attempts to directly maximize performance, while PPO indirectly maximizes performance using a surrogate objective function (OpenAI).

Methodology

The experiments for this project were run entirely in Google Collab using Python. The pretrained PPO and A2C models I used were found from HuggingFace, with the original algorithms being from the StableBaselines3 library. The LunarLander-v3 environment this project runs in is from OpenAI Gym. I additionally used the Optuna library for automating the hyperparameter tuning process and TensorBoard for creating graphs of my output for viewing.

The goal of this project was to hypertune the parameters of the pretrained PPO and A2C algorithms; that is, adjusting the hyperparameters of each algorithm, such as the learning rate which governs how influential each run is to the overall AI or entropy which determines how random the output of the AI is, to create the most optimal algorithm for playing the LunarLander game.

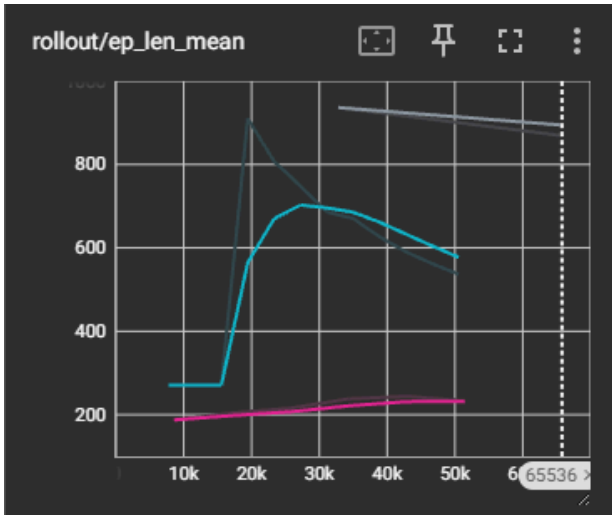
Aidan Beesley
CIS 730: Principles of Artificial Intelligence

The process of assembling the hyperparameters consisted of using Optuna's suggest_int and _float methods to automatically tune four hyperparameters, picking from a predetermined input range for 10 trials. The below hyperparameters are what was received from Optuna, along with the baseline PPO hyperparameters from the HuggingFace repo. Of note is that the HuggingFace repo does not specify the learning rate of the baseline model, so I can only assume it is the baseline of the original PPO model from StableBaseline3's documentation.

	Baseline (HuggingFace, SB3)	PPO Tuned	A2C Tuned
gamma	0.999	0.9937781866542922	0.902804073521881
learning_rate	0.003*	0.00022529844732911952	0.0029056840265277825
n_steps	1024	243	536
ent_coef	0.01	0.00019070775498326704	9.365240201733659e-06

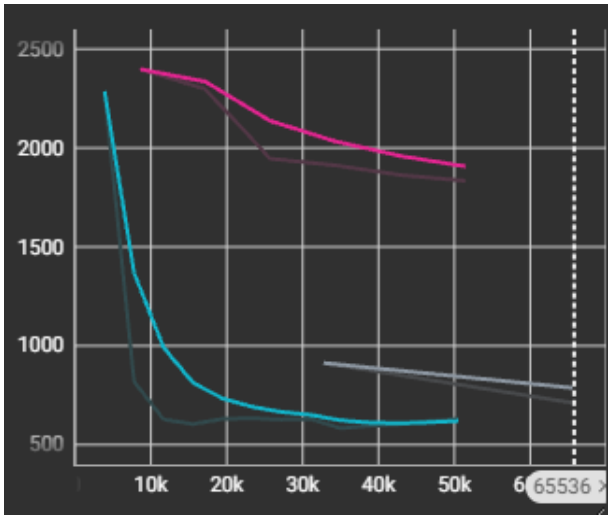
Experimental Design and Results

Using the two metrics of accuracy and speed of execution, these were the initial results I received after training the AIs for 50,000 steps, with the PPO baseline model in grey, PPO tuned model in blue, and A2C model in pink.

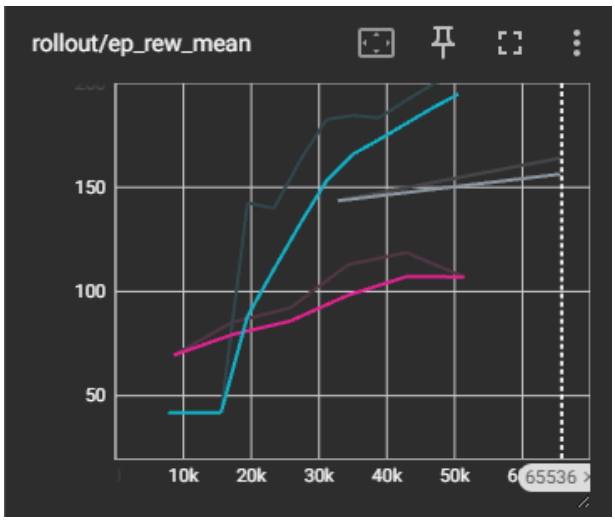


The above graph is the measure of the mean length of each individual episode of training, rather than the algorithm as a whole. In the context of LunarLander-v3, episode

length would be the amount of time it takes each algorithm to land the lander, success or failure.



This second graph, meanwhile, is a measure of the physical time it took to run each algorithm, measured in frames per second. Using the above two algorithms, we can conclude that the baseline PPO model remained mostly static, which is to be expected given that it isn't changing at all. The tuned PPO algorithm's training went very well, as the sharp spike in episode length before falling off along with the time it takes to run the algorithm decreasing exponentially indicates that it is taking shorter time both in episodes and in absolute time to achieve the optimal reward and that further training would likely improve the former. The A2C model, meanwhile, has comparatively little change in the episode length while decreasing absolute time to run the algorithm, indicating that it is at least getting faster.



The above is a graph of the episode reward mean from the training, and is the primary metric for measuring the success of each algorithm. The PPO tuned model has far and away the greatest reward, while A2C has a much slower rise below the PPO baseline model that continues to remain mostly flat. As for why this may be, I believe that A2C's performance can likely be explained by A2C being an initial worse algorithm, thus starting at a lower point than either PPO algorithms. As for why my tuned PPO model has a higher reward, that is likely an indication of the hyperparameter tuning's success.

After these algorithms were trained, they were tested using the built-in evaluate_policy method from StableBaselines3, using the same randomly seeded environment for each algorithm where the number of evaluations per environment is 20, taking an average of 10 runs. This was all done to ensure that a large sample size of each algorithm were taken, so that one algorithm wouldn't get particularly lucky or unlucky with an environment that was well suited to it, but poorly suited to others.

	Avg. Mean Reward	Comparison to Baseline
Baseline	228.56	~
PPO	244.67	+16.11
A2C	163.93	-64.63

As can be seen above, a similar result from the initial training graph is shown, with the tuned PPO model doing better than the baseline, which is better than the A2C model, though the increase in average mean reward from the baseline to tuned model is not as sharp as earlier. A2C's place behind both PPO models has already been discussed, though the tuned PPO model's comparatively lesser success is harder to explain. Ultimately, I believe it is likely due to the pretrained model already being well tuned and more thorough testing done compared to the one testing run that was done.

Summary and Future Work

In summary, my experiment in hyperparameter tuning was able to produce a PPO model that was able to improve on an existing pretrained model. For future work, my first priority would be to test more algorithms than A2C and PPO. My initial plan was to test PPO and DQN, but due to technical issues with DQN I could not get that algorithm to work. Another simple route for future work I would like to take is allowing for more training and testing time for all of my algorithms. 50,000 steps is not a small amount, but I believe that more time would still allow for improvements to both algorithms. Additionally, I would also like to test the difference between the LunarLander-v2 and -v3

environments, as while I do not believe there would be any major difference between them, the algorithms I used were pretrained on the -v2 environment which may affect their performance. Lastly, I would like to replicate this experiment on other environments in Gym, such as CartPole, to see how the different algorithms may react to different environments.

References

OpenAI. (n.d.). *Part 2: Kinds of RL Algorithms—Spinning Up documentation*. Retrieved May 14, 2025, from https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html

SB3. (n.d.). *PPO -- StableBaslines3.2.6.1a1 documentation*. Retrieved May 14, 2025, from https://stable-baselines3.readthedocs.io/en/master/modules.ppo.html

HuggingFace. (n.d.). *Sb3/ppo-LunarLander-v2 · Hugging Face.* Retrieved May 14, 2025, from https://huggingface.co/sb3/ppo-LunarLander-v2