Jacob Legg, Chris Loura, Sicheng Chen

CIS 730: Principles of Artificial Intelligence

Dr. William Hsu

May 5, 2025

Hyperparameter Tuning of RL Algorithms - Final Project Proposal

Introduction

For this project, we planned to use four reinforcement learning algorithms that may complete the gymnasium MountainCar-v0 environment using a Deep Q-Learning Network (DQN) as a baseline and Trust Region Policy Optimization (TRPO), Proximal Policy Optimization (PPO), and Advantage Actor-Critic (A2C) as the other three algorithms. The goal was to fine tune the hyperparameters for each of the models and discover how the tuning improves upon the models and how the models compare to each other. We switched away from Bipedal Walker because we want to focus more on hyperparameter tuning and model comparison instead of input cleaning. This project was done by Jacob Legg, Christopher Loura and Sicheng Chen.

For the project, we used a variety of tools and libraries. For the environment we used OpenAI Gymnasium's MountainCar-v0. For the algorithms, we used StableBaselines3's versions of DQN, A2C, TRPO and PPO. We used Optuna for hyperparameter tuning the algorithms, and finally, this was all integrated into Google Colab to run on their CPUs.

Background and Related Work

For this project, we decided upon four RL algorithms to train - DQN, A2C, TRPO and PPO.

DQN or Deep Q-Network is an off-policy learning algorithm that uses Epsilon-Greedy to select the action with the highest state-action value (Lisi, 2021). The algorithm either chooses a random action with probability ε or it takes the action with the highest Q-value. This is called exploration and exploitation respectively. DQN's epsilon value was also shrunk in order to eventually switch from mostly exploration to mostly exploitation. The algorithm utilizes two neural networks - prediction and target - where prediction gets updated more frequently than target to reduce the non-stationarity target problem (n.d., Hugging Face).

DQN serves as our baseline since generally, it underperforms the other algorithms by a significant margin while also being able to at least produce some results.

A2C, or Advantage Actor Critic, is a policy-based Actor-Critic algorithm. A2C uses an advantage function as the critic. The advantage function is what calculates how better taking that action at a state is compared to the average value of the state (Simonini, 2022). The advantage is used to directly calculate the policy update. As such, this results in a more stable learning than standard Actor-Critic methods which use an action-value function. Finally, to estimate the value of the advantage function, TD error can be calculated instead.

TRPO or Trust Region Policy Optimization is a policy-based Actor-Critic algorithm. It is a policy-gradient algorithm, which means that it evaluates and directly improves the same policy that selects actions (Lisi, 2021). It maximizes surrogate advantage to obtain the most optimal policy. KL divergence, the probability distribution difference between new and old policies, is restricted by the trust region, δ, to limit major policy updates and risky exploration (OpenAI 2018).

PPO, or Proximal Policy Optimization is another policy-based Actor-Critic algorithm. Because PPO is the successor of TRPO, it uses the policy-gradient method as well. Its goal is to only make modest updates to the policy and avoid large policy updates. In order to do that, PPO uses a ratio that indicates a difference between the old policy and the current policy (n.d., Hugging Face). This ratio is then clipped from a range of $[1-\varepsilon, 1+\varepsilon]$ to ensure training stability and small policy updates. This results in a simpler calculation than limiting the KL-divergence by δ.

For sampling, the Tree-structured Parzen Estimator was used over other samplers because of our large parameter space. Rather than relying on an exhaustive method of Grid Search and randomization of Random Search, it benefits us the most to get educated guesses on which parameters are the most significant through Gaussian Estimator and modify them respectively.

Methodology

While Hyperparameter tuning for DQN, PPO, A2C and TRPO is a well–established area with extensive research and literature, we chose to approach this challenge on our own. Rather than relying on published optimal values or tuning recommendations, we conducted our own research and performed our own exploration of the hyperparameter space for each algorithm. This methodological approach allowed us to develop a deeper understanding of how each hyperparameter affects the performance of each algorithm. This independent research into hyperparameter tuning represented a commitment to empirical investigation rather than simply implementing established best hyperparameters and their values from existing sources.

Experimental Design

MountainCar-v0 was used mainly because of its unique reward function. Most RL environments try to maximize their rewards by obtaining as much reward as possible. However, with mountain car, the agent is trying to maximize the amount of reward by minimizing the amount of reward lost. We attempted to achieve the best scores with the lowest variance by tuning hyperparameters along with the reward function and quantization.

In order to improve the efficiency of the algorithms, we quantized the environment's observation space. To figure out which quantization to use to run our hyperparameter tuning on, we ran 15 tests for each algorithm over the following quantizations: (5, 5), (10, 10), (15, 15), (20, 20), (25, 25), (30, 30). This quantizing was done by hand and we continued to increase the quantizations until the mean reward began to level out for every algorithm. Since position and velocity are spatial observations, we decided to lock the ratio between the two for this testing, hence the choices above. We discovered that the best quantization was (20, 20), and therefore used this quantization for fine tuning. We did have good results with a quantization of (15, 15), but the mean reward had a much higher variance with (15, 15) and underperformed (20, 20) by a small margin.

We also attempted to implement a custom reward function that would be better than the mountain car's default. This was done to attempt to reduce mean episode length. We ran tests on three reward functions - the default reward function, a reward function developed by MehdiShahbazi and a custom reward function developed by Christopher and Jacob. MehdiShahbazi's reward function heavily prioritizes rightward movement of the agent. Chris's and Jacob's reward function prioritizes both rightward movement and velocity by keeping track of the agent's maximum x position and velocity. If the agent beats the maximum, it gets more reward. There is also reward obtained when the car's position is a greater distance away from the spawn position while also applying a higher velocity.

For model evaluation, each algorithm, with and without hyperparameters, was trained and evaluated at least 20 times with 5000 episodes per training with 100 evaluation episodes. After the evaluations we would record the tensorboard data, mean reward (N=100), and mean episode reward over time graph for the best model of all the trained models. We would also record the number of models that successfully reach the flag during the 20 training sessions during evaluation in order to check the variance of the models during training.

Hand optimization and tools such as Optuna were used to optimize hyperparameters of the model for further comparison. For each algorithm, we performed 100 optimization trials and 5 startup trials with 2,500 episodes per trial, where their performance would be measured by the mean reward of 100 evaluation episodes. Pruning trials will start after 25 trials and hyperparameters from the best trial were selected and used for the tuned test.

After conducting our own research and testing, we chose the following hyperparameters to tune for each algorithm:

Figure 1

Hyperparameters Chosen for Tuning

DQN	**A2C**	**TRPO**	**PPO**
gamma, tau, learning_starts, learning_rate, train_freq, max_grad_norm, replay_buffer_class, buffer_size, exploration_initial_eps, exploration_final_eps	gamma, max_grad_norm, gae_lambda, n_steps, learning_rater, vf_coef, ent_coef, rms_prop_ps, use_rms_prop	learning_rate, gamma, cg_max_steps, cg_damping, line, search_shrinking_factor, line_search_max_iter, n_critic_updates, gae_lambda, target_kl, sub_sampling_factor	gamma, max_grad_norm, gae_lambda, learning_rate, vf_coef, ent_coef, clip_range, n_epochs

Informal Results

After running DQN and PPO against all of these reward functions 10 times each, we concluded that the default mountain car environment reward performs the best. We believe that this is due to the environment's sensitivity and a lack of thorough exploration of every possible reward feature.

PPO with the default reward function showed successful results on average 1 in every 7 untuned trials, whereas with tuned hyperparameters, it showed successful results every run. DQN obtained good results on average 1 in every 15 untuned trials. DQN with tuned hyperparameters was able to show good results 1 in every 2 runs. With the best scores for each being shown in Figure 2.

Both TRPO and A2C were unable to converge on an answer. We hypothesize that this is the case because of how they calculate policy gradients, which can cause instability in policy updates and severely hinder exploration. TRPO's policy gradient calculation uses hard constraints on top of complex calculations. A2C's calculation involves directly using the advantage function with no constraints on policy update, which thus limits learning and leads to repeated exploration.

Overall, tuned and untuned PPO are shown to have the best performance among the algorithms tested. Although untuned PPO had a mean reward of -104 at its best, tuned PPO showed more consistent and successful results. Tuned and untuned DQN showed second best performance with a mean reward of -118.48 and -110 respectively. A2C and TRPO showed expected mean reward of -200 consistently. See Figure 2 for the numerical visualization.

The final hyperparameter values that were discovered through this project can be found in Appendix A.

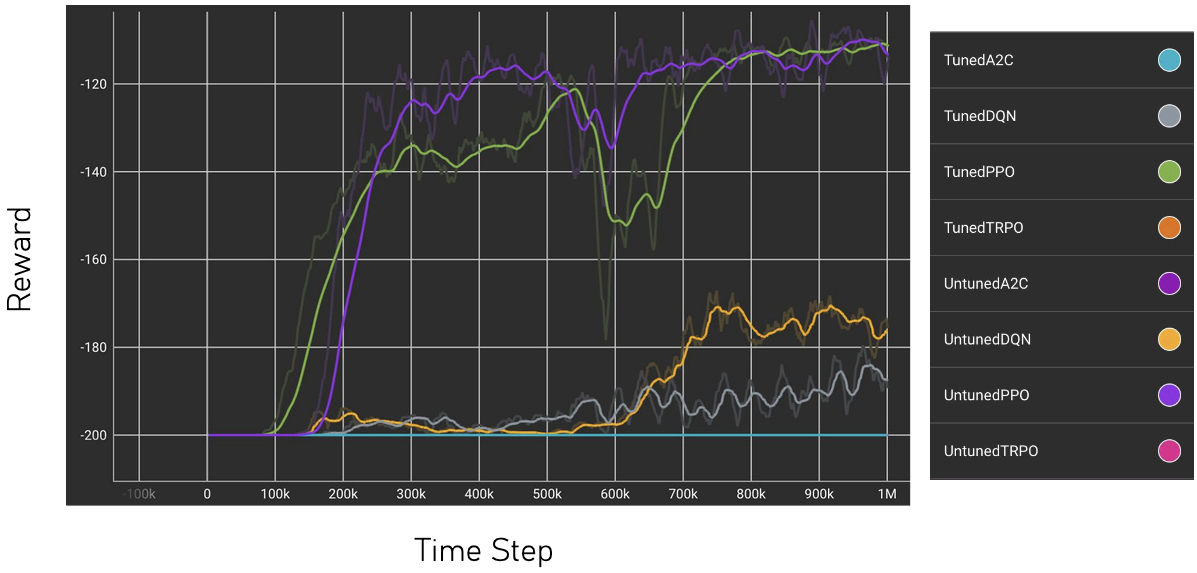
Figure 2

Mean Reward for best runs for each algorithm (N=100)

	Untuned	Tuned
DQN	-110 +/- didn't record	-118.48 +/- 16.35
PPO	-104 +/- 6.7	-116.62 +/- didn't record
A2C	-200 +/- 0	-200 +/- 0
TRPO	-200 +/- 0	-200 +/- 0

Figure 3

Episode Mean Reward over a million time steps



This graph of the episode mean reward further demonstrates the conclusions drawn above. We saw both Untuned A2C and Untuned TRPO flatline at -200 mean reward. Meanwhile, Untuned PPO only slightly underperformed Tuned PPO largely due to the amount of variance that was observed with PPO. Tuned DQN in this graph also underperformed Untuned DQN, which is another example of the amount of variance discovered in the tests. However, though it is not visible in either Figure 2 or Figure 3, the most interesting observation, when viewing videos of the agents, is that Tuned TRPO and Tuned A2C began to learn good behaviors despite the fact that they did not reach the goal. If more time was given for training, the two tuned algorithms could potentially converge on a model.

Summary

Running the tests with and without hyperparameter optimization, we concluded that untuned PPO and tuned PPO had the better performance compared to both tuned and untuned DQN. These two algorithms out performed both tuned and untuned versions of A2C and TRPO. Both TRPO and A2C were unable to converge on an answer as stated above. We hypothesize the calculation of policy gradients and the sensitive nature of the environment, which likely caused instability in policy updates and severely hindered exploration. TRPO's policy uses hard constants on top of complex calculations. A2C's calculation involves directly using the advantage function with no constraints on policy update, which thus limits learning and exploration.

Overall, both DQN and PPO were able to find decent solutions with significant results. The tuned versions of each were also able to find consistent results, as opposed to their

untuned counterparts. Unfortunately, both the tuned and untuned versions of A2C and TRPO were unable to find both significant results or any solution. However, when tuned, both A2C and TRPO began to converge on an answer and learn better behaviors.

Analysing the environment, we can see that MountainCar-v0 is incredibly sensitive to actions and is limited in observations. The success of the algorithms was dependent primarily on the seed that was used, which would also determine its path for initial exploration. This limited the effectiveness of both fine tuning and the testability of the algorithms. The environment also made it very difficult to create a custom reward function that wouldn't be abused or that would overly punish the agent due to the limited and sensitive observation space.

Future Work

Per the recommendation of Dr. Hsu, we would like to try Q-Learning approaches such as Impala and IDAAC to assess their effectiveness in the mountain car environment. We would also like to experiment with more on-policy approaches such as REINFORCE, TD3, SAC, and DDPG in a continuous mountain car environment and see how they compare to the four algorithms and each other. It would also be interesting to create a balanced custom reward function in order to reduce mean episode length.

In terms of hyperparameter tuning, more work needs to be done on the evaluation of hyperparameter importance. Optuna has a built-in visualization library that displays a bar graph containing the most important hyperparameters. It would be interesting to see if we could fine-tune every hyperparameter for every algorithm, but also get the graph of the most important hyperparameters for every algorithm. Ideally, this tuning for every hyperparameter would be tuned for a longer amount of time instead of the default 12 hour limit allowed by Google Colab. It would be interesting to run these larger tests on a supercomputer like Beocat or even on a stronger CPU/GPU.

Finally, we would like to do a more formal comparison between all of the models by using various statistical analysis methods. Some of these would include dual p-values and an error bar comparison.

Bibliography

A2C — Stable Baselines3 2.2.0a7 documentation. (n.d.). Stable-Baselines3.Readthedocs.io. https://stable-baselines3.readthedocs.io/en/master/modules/a2c.html

araffin. (2024, July 11). GitHub - araffin/sbx: SBX: Stable Baselines Jax (SB3 + Jax). GitHub. https://github.com/araffin/sbx?tab=readme-ov-file

ashishraste. (2019). openai-gyms/cartpole-v0/qcartpole_agent.py at master · ashishraste/openai-gyms. GitHub. https://github.com/ashishraste/openai-gyms/blob/master/cartpole-v0/qcartpole_agent.py

Crazymuse. (2018, April 24). TRPO (Trust Region Policy Optimization) : In depth Research Paper Review. YouTube. https://www.youtube.com/watch?v=CKaN5PgkSBc

DQN — Stable Baselines3 2.6.1a0 documentation. (2021). Readthedocs.io. https://stable-baselines3.readthedocs.io/en/master/modules/dqn.html#module-stable_baselines3.dqn

Elvers, A. (2019, November 22). File:Hyperparameter Optimization using Tree-Structured Parzen Estimators.svg - Wikimedia Commons. Wikimedia.org. https://commons.wikimedia.org/wiki/File:Hyperparameter_Optimization_using_Tree-Structured_Parzen_Estimators.svg

Gymnasium Documentation. (n.d.). Gymnasium.farama.org. https://gymnasium.farama.org/environments/classic_control/mountain_car/

Horgan, C. (2023, April 4). Building a Tree-Structured Parzen Estimator from Scratch (Kind Of) | Towards Data Science. Towards Data Science. https://towardsdatascience.com/building-a-tree-structured-parzen-estimator-from-scratch-kind-of-20ed31770478/

Hui, J. (2018, October 21). RL — Trust Region Policy Optimization (TRPO) Explained. Medium. https://jonathan-hui.medium.com/rl-trust-region-policy-optimization-trpo-explained-a6ee04eeeee9

Lisi, A. (2021, February 24). Beating Pong using Reinforcement Learning — Part 2 A2C and PPO. Analytics Vidhya. https://medium.com/analytics-vidhya/beating-pong-using-reinforcement-learning-part-2-a2c-and-ppo-b83391dd3657

Nair, A., Srinivasan, P., Blackwell, S., Alcicek, C., Fearon, R., Maria, D., Panneershelvam, V., Suleyman, M., Beattie, C., Petersen, S., Legg, S., Mnih, V., Kavukcuoglu, K., & Silver, D. (2015). Massively Parallel Methods for Deep Reinforcement Learning. ArXiv.org. https://arxiv.org/abs/1507.04296

optuna. (2021). optuna-examples/rl/sb3_simple.py at main · optuna/optuna-examples. GitHub. https://github.com/optuna/optuna-examples/blob/main/rl/sb3_simple.py

Optuna: A hyperparameter optimization framework — Optuna 2.10.1 documentation. (n.d.). Optuna.readthedocs.io. https://optuna.readthedocs.io/en/stable/

Porcher, F. (2023, July 17). Quantization (A Guide for complete beginners) - AI Mind. Medium; AI Mind. https://pub.aimind.so/quantization-guide-for-complete-beginners-ac4555cf295f

PPO — Stable Baselines3 1.4.1a3 documentation. (n.d.). Stable-Baselines3.Readthedocs.io. https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html

PPO Algorithm Hyperparameters Tuning | Restackio. (2025). Restack.io. https://www.restack.io/p/hyperparameter-tuning-answer-ppo-algorithm-hyperparameters-cat-ai

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., & Abbeel, P. (2015). Trust Region Policy Optimization. ArXiv.org. https://arxiv.org/abs/1502.05477

Simonini, T. (2022a). Advantage Actor Critic (A2C). Huggingface.co. https://huggingface.co/blog/deep-rl-a2c#advantage-actor-critic-a2c

Simonini, T. (2022b, August 5). Proximal Policy Optimization (PPO). Huggingface.co. https://huggingface.co/blog/deep-rl-ppo

Speaker, O. N. (2024, November 10). Reinforcement Learning: Scaling Up with A2C — Hyperparameter Tuning. Medium. https://medium.com/@old.noisy.speaker/reinforcement-learning-scaling-up-with-a2c-hyperparameter-tuning-43864ba1f299

The Deep Q-Network (DQN) - Hugging Face Deep RL Course. (2025). Huggingface.co. https://huggingface.co/learn/deep-rl-course/en/unit3/deep-q-network

TRPO — Stable Baselines3 - Contrib 2.6.0a2 documentation. (2021). Readthedocs.io. https://sb3-contrib.readthedocs.io/en/master/modules/trpo.html

哲熙. (2024, September 23). DQN for Mountain Car RL Training - 哲熙 - Medium. Medium. https://medium.com/@310274movie/dqn-for-mountain-car-rl-training-3b169a264534

Appendix A

Tuned Hyperparameters for DQN: Value of -138.76

Gamma	0.92193641832

Max Grad Norm	0.9003823121616462
Tau	0.23178397221657238
Learning Rate	2.3110058432045783e-05
Learning Starts	583
Training Frequency	76
Replay Buffer	Replay
Exploration Initial Epsilon	0.5605956292015678
Exploration Final Epsilon	0.32310556739485297
Buffer Size	34461

Tuned Hyperparameters for PPO: Value of -116.62

Gamma	0.99962351145
Max Grad Norm	0.9558861968131098
Gae Lambda	0.93732555426
Learning Rate	0.0008036518212978304
VF Coefficient	0.17925670781108438
Entropy Coefficient	0.035073620842585695
Number of Epochs	12
Clip Range	0.7344958587946594

Tuned Hyperparameters for A2C: Value of -177.6

Gamma	0.99865489002
Max Grad Norm	0.866881944779409

Gae Lambda	0.99779588013
N Step Exp	8
Learning Rate	0.00014873761184142873
VF Coefficient	0.08195608622843578
Entropy Coefficient	0.080037004469273
RMS Prop Eps	6.003951202932881e-07
Use RMS Prop	True

Tuned Hyperparameters TRPO

Learning Rate	0.0004299052180398058
Gamma	0.0602720394278056
CG Max Steps	7
CG Damping	0.010753382215110234
Line Search Shrinking Factor	0.09536592655556136
Line Search Max Iter	8
n Critic Updates	2
Gae Lambda	0.006161502929248017
Target KL	0.006851390877584308
Sub Sampling Factor	7