May 2021

Solving LunarLander-v2 through Hyperparameter Tuning

Running the LunarLander with the original hyperparameters ("default") resulted in a model that displayed a strong random element (showing signs of learning through improving reward, before dropping back significantly) as well as high volatility; during the 140 iterations that were run exemplary, the lander crashes and does not land consistently inside the flag poles or close as required. In order to smoothly land the LunarLander in between the two flagpoles, it is necessary to train the LunaLander to learn fast and to achieve a more consistent performance.

Parameter	Default	Optimized
SCALE_REWARD	0.01	0.1
MIN_REWARD	-1000	-500
HIDDEN_SIZE	128	128
BATCH_SIZE	512	256
DISCOUNT	0.99	0.99
GAE_LAMBDA	0.95	0.95
PPO_CLIP	0.2	0.1
PPO_EPOCHS	10	20
MAX_GRAD_NORM	1	1
ENTROPY_FACTOR	0	0
ACTOR_LEARNING_RATE	1.00E-04	1.00E-04
CRITIC_LEARNING_RATE	1.00E-04	1.00E-04
RECURRENT_SEQ_LEN	8	8
RECURRENT_LAYERS	1	1
ROLLOUT_STEPS	2048	500
PARALLEL_ROLLOUTS	8	8
PATIENCE	200	200
TRAINABLE_STD_DEV	False	False
INIT_LOG_STD_DEV	0	0

Figure 1: Final Hyperparameters

Based on some initial research, I identified key hyperparameters to influence model performance and started adjusting them one by one while keeping default parameters constant to assess their impact on model performance before finally compiling those findings for further fine-tuning. I identified the following key parameters with the biggest impact on model performance:

(1) How much experience should the agent gather before updating the policy

The algorithm gathers information up to the maximum number of steps defined (ROLLOUT_ STEPS) before performing SGD update on the batch of gathered trials (BATCH_SIZE) for the specified number of epochs (PPO_EPOCS).

- **BATCH_SIZE:** Batch size refers to the amount of information the model is trained on before updating the model's weights. Testing different batch sizes (4000, 512, 256, 128, 10) didn't yield any significant changes in performance (mean reward or entropy), in theory smaller batch size should help the model to better generalize since it does see smaller parts of the entire data at once.
- **PPO_EPOCHS:** At the same time, the number of epochs was moderately increased (to 20) thereby allowing the model to see more data at once. However, it also increases training time (tested vs. 10, 25, 30). Batch size and the number of epochs are closely correlated and should be adapted in parallel, the optimal combination could for example be found using gridsearch.
- **ROLLOUT_STEPS:** Reducing the step size reduces complexity and training time, while at the same time the number of steps should not oversimplify the problem (danger of resulting in erratic behavior). I achieved best results with reducing the number of steps to 500 (tested vs. 2048, 250)

(2) How to update the policy

If the policy is updated with the newly learned information too rapidly, policy performance can collapse drastically and never recover. PPO uses a surrogate loss function with a clipping factor to avoid too large updates and thus prevents the training from becoming too unstable and volatile.

- **PPO_CLIP:** Best results were achieved with a clipping of 0.1 (tested vs. 0.0, 0.15, 0.2, 0.3)
- **DISCOUNT:** represents how much weight is given to future rewards, a value of 0 values immediate rewards vs a value of 0.9 represents a large weight given to rewards in the distant future. I kept the default value of 0.99 since trying a discount factor of 0.9 resulted in degrading performance, similarly a discount factor of 0.999, which was encouraging to reward all steps.

- **ENTROPY_FACTOR:** it acts as a regularizer that encourages the model to explore different policies, thus introducing randomness to the model in choosing its next action. Introducing an entropy factor (0.01) positively impacted mean reward.

(3) How is the agent learning:

- **LEARNING_RATE:** trying with different learning rates, I found a learning rate of 1E-4 to provide the best ratio between learning speed (large enough to quickly converge) while not too high and jump around the optimum (never converge). No distinction was made between actor and critic learning rate; however, I'd recommend to assess this in a next step.
- **SCALE_REWARD:** Large and sparse rewards can lead to saturation and inefficiency; research showed that reward scaling was effective in mitigating these effects and improved performance. Experimenting with the scale reward (0.01, 0.1, 1, 10) showed that there was a slight performance improvement when increasing the reward scaling, however at increasing values the mean reward converged very quickly to an optimum before quickly deteriorating in performance. I therefore decided to use a slightly higher scale reward of 0.1 (vs. default 0.01) but further tests necessary
- **MIN_REWARD:** In relation to the positive rewards (100-140 points for moving from top to landing pad plus max. additional +100 points for finishing plus leg contaxt +2x10 + 200 for solved), the maximum negative reward of -1000 seemed very high. Using a reward of -500 resulted in improved performance and faster convergence vs a reward of -1000.

Comment: while I believe that reward hyperparameters are very crucial in the optimization, I see a risk of incentivizing unwanted behaviors when adjusting these parameters without proper understanding or testing results until full convergence!

Conclusion

Using these hyperparameters, the model resulted in consistent positive results after ca. 150 iterations and further improved (trained until 270 iterations). Nevertheless, additional training and optimization could be done, especially:

- Due to limits in time and GPU support, results were obtained based on a very small number of iterations (between 100-200 iterations except for final model with 270 iterations, 14 experiments ran, with 20-50 seconds per iteration). I'd suggest resuming training with more time and GPU
- After an initial decrease in the beginning, mean entropy ("how predictable are the actions of the agent) did not further decrease and finally started increasing again – this suggests that entropy should be further optimized
- Several parameters were kept unchanged, some might further positively improve performance

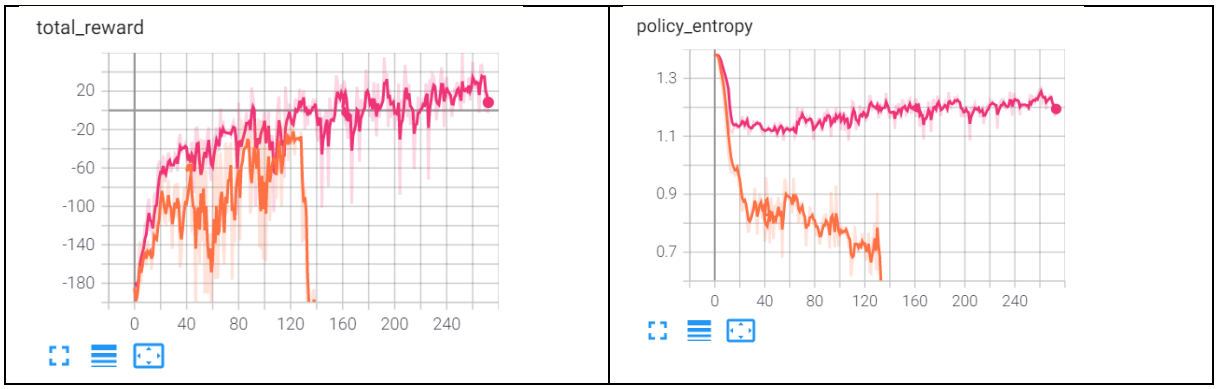

Figure 2: Performance Default values (orange) vs. best-performing hyperparameters (pink) *With Smoothing 93% (TensorBoard)*