National Yang Ming Chiao Tung University (國立陽明交通大學)

Reinforcement Learning HW2

Quentin Ducoulombier (昆丁)
Student Id: 312551811

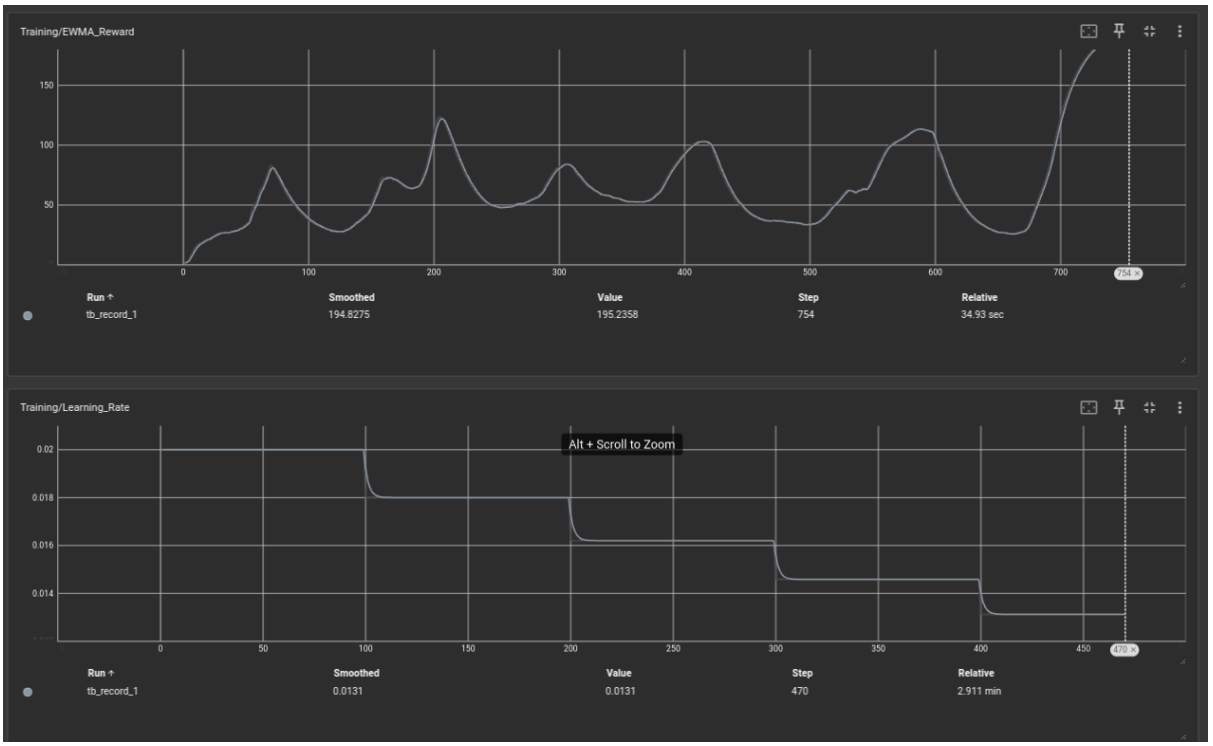
Reinforcement Learning

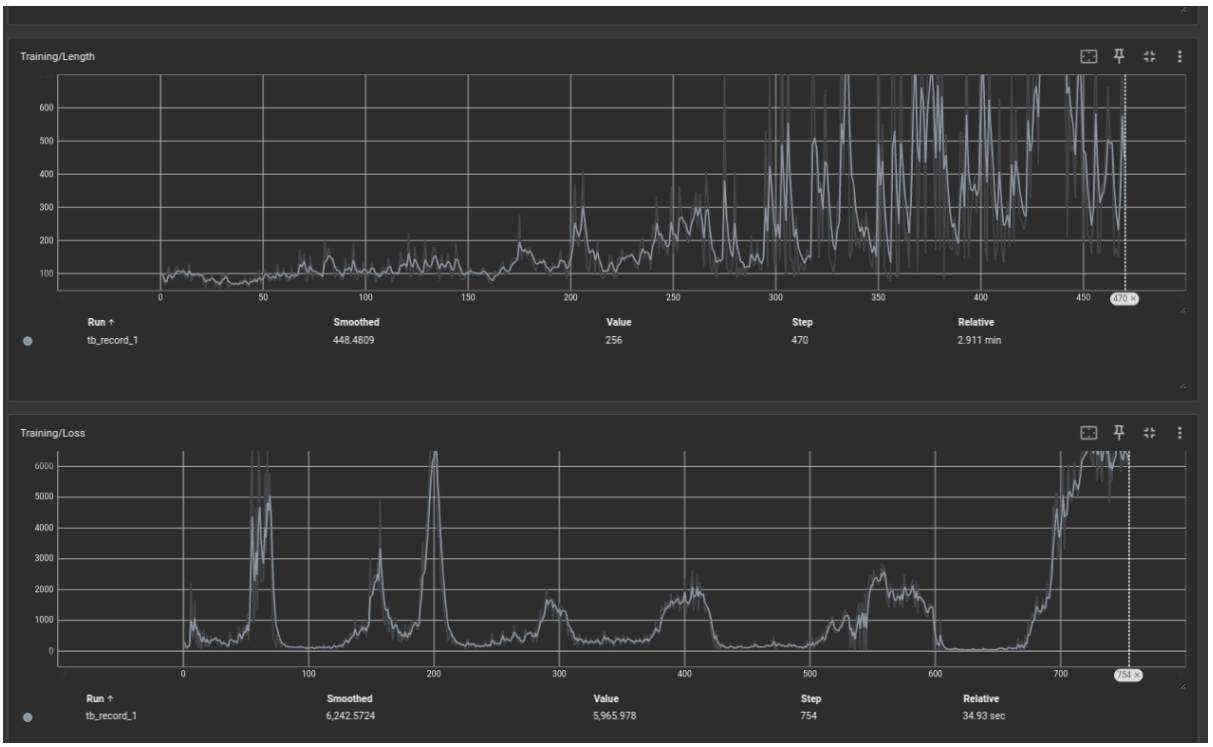
謝秉均

2024-04-19

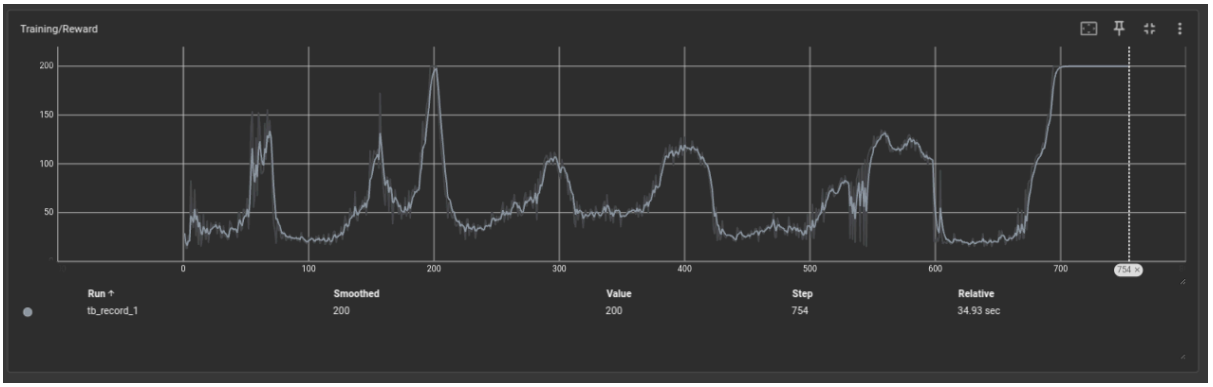
Reinforce vanilla

In my experiment implementing the Vanilla REINFORCE algorithm on the "CartPole-v0" environment using PyTorch, the model successfully learned to balance the pole on the cart, as indicated by increasing rewards and episode lengths. The policy network architecture consisted of an input layer matching the environment's observation space of four, a hidden layer with 128 neurons using ReLU activation, and an output layer for two possible actions using softmax activation. I used a learning rate of 0.01, a discount factor of 0.999, and the Adam optimizer for training. Adjustments to these hyperparameters, especially the learning rate, significantly affected the learning stability and speed. Changes often led to divergences or significantly slower learning processes, highlighting the sensitivity of the REINFORCE algorithm to hyperparameter settings. This experiment underscored the importance of careful tuning and consideration of the learning environment's specifics for efficient training.




Training/Length
600
500
400
300
200
100
0
50
100
150
200
250
300
350
400
450
470
Run ↑
Smoothed
Value
Step
Relative
tb_record_1
448.4809
256
470
2.911 min

Training/Loss
6000
5000
4000
3000
2000
1000
0
0
100
200
300
400
500
600
700
754
Run ↑
Smoothed
Value
Step
Relative
tb_record_1
6,242.5724
5,965.978
754
34.93 sec


Training/Reward
200
150
100
50
0
100
200
300
400
500
600
700
754
Run ↑
Smoothed
Value
Step
Relative
tb_record_1
200
200
754
34.93 sec

Reinforce baseline

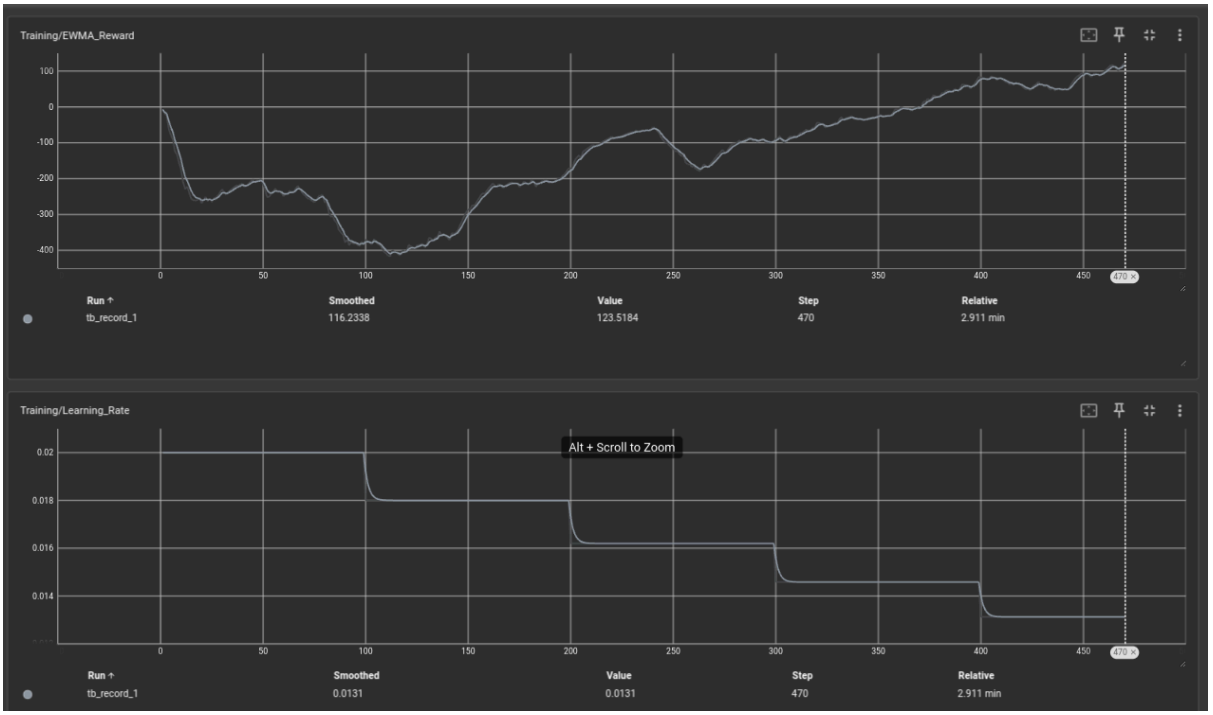
The model demonstrated varying performance based on adjustments to the learning rate (lr). Initially set at 0.01, it required 647 episodes to reach a satisfactory performance level. Altering the learning rate provided mixed results:

- Lowering to 0.001 led to excessively long training times, making it impractical.
- Increasing to 0.02 optimized performance, reducing the number of episodes to 470 to achieve similar rewards.
- Small adjustments around 0.01 (like 0.009 and 0.019) generally increased the number of episodes required, with diminishing returns as the rate approached 0.03.

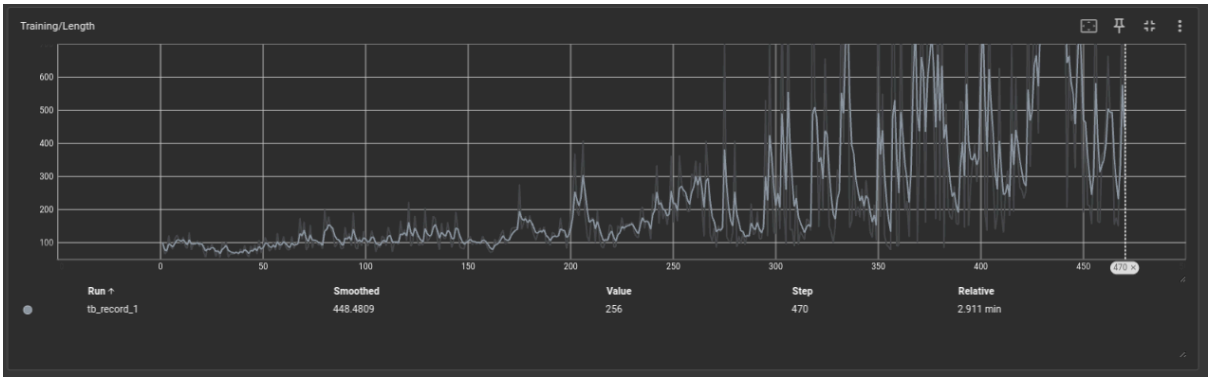
The optimal learning rate was found to be 0.02, balancing efficiency and convergence speed. Adjustments to other hyperparameters like gamma were also tested; however, a gamma of 0.99 proved most effective, with decreases leading to poorer outcomes.

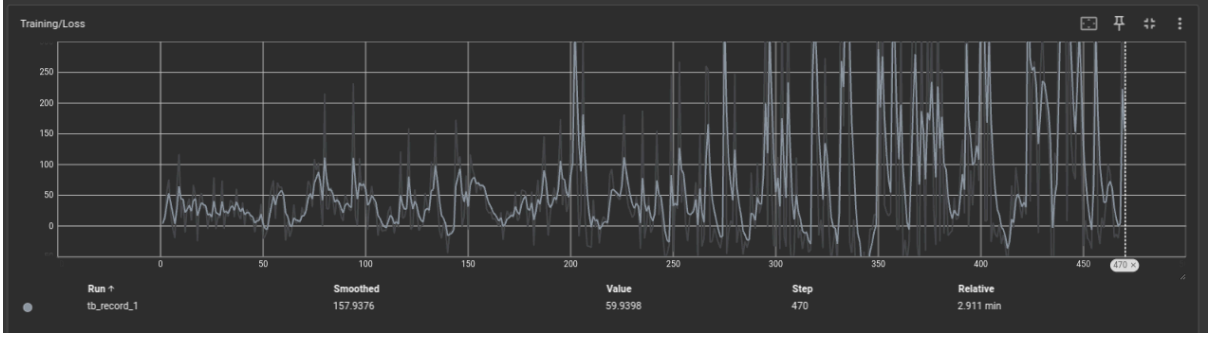
Hyperparameters Documented:

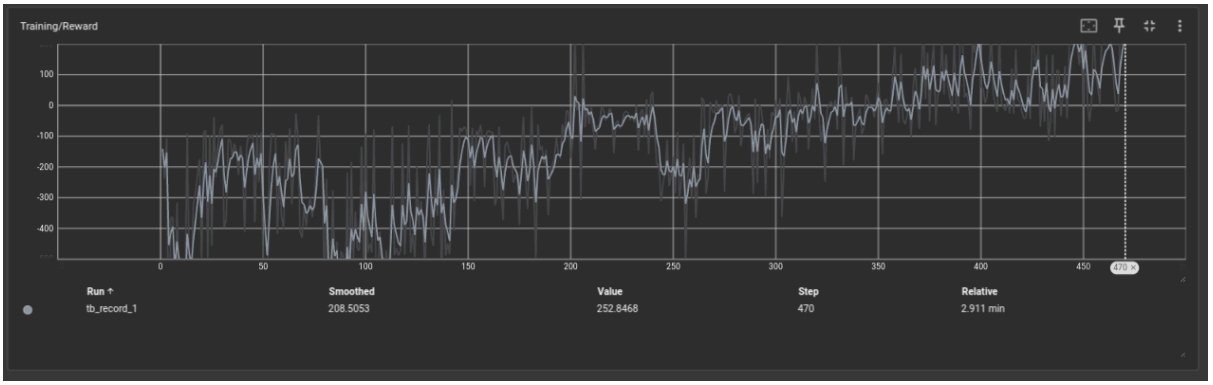
- **Learning Rate**: Varied in experiments; optimal at 0.02.
- **Gamma (Discount Factor)**: Maintained at 0.99, tested down to 0.95 and 0.98 with suboptimal results.
- **Network Architecture**:
    - Input Layer: Matches the observation space of Lunar Lander (state size).
    - Shared Layer: 128 hidden units with ReLU activation.
    - Action Layer: Outputs action probabilities using softmax.
    - Value Layer: Outputs a single value estimate.
- **Optimizer**: Adam, suitable for its adaptive learning rate capabilities.
- **Scheduler**: StepLR, reducing the learning rate by 10% every 100 episodes to adapt to changing gradients.


Training/EWMA_Reward
100
0
-100
-200
-300
-400
0
50
100
150
200
250
300
350
400
450
470
Run
Smoothed
Value
Step
Relative
tb_record_1
116.2338
123.5184
470
2.911 min

Training/Learning_Rate
Alt + Scroll to Zoom
0.02
0.018
0.016
0.014
0
50
100
150
200
250
300
350
400
450
470
Run
Smoothed
Value
Step
Relative
tb_record_1
0.0131
0.0131
470
2.911 min


Training/Length
600
500
400
300
200
100
0
50
100
150
200
250
300
350
400
450
470
Run
Smoothed
Value
Step
Relative
tb_record_1
448.4809
256
470
2.911 min


Training/Loss
250
200
150
100
50
0
0
50
100
150
200
250
300
350
400
450
470
Run
Smoothed
Value
Step
Relative
tb_record_1
157.9376
59.9398
470
2.911 min


Training/Reward
100
0
-100
-200
-300
-400
0
50
100
150
200
250
300
350
400
450
470
Run
Smoothed
Value
Step
Relative
tb_record_1
208.5053
252.8468
470
2.911 min

Reinforce with GAE

Generalized Advantage Estimation (GAE) Implementation:

Initialization

The `GAE` class is initialized with the parameters `gamma` for discounting future rewards, and `lambda_` for smoothing the advantage estimates over multiple timesteps. These parameters control how the future rewards influence the present value estimates.

Calculation Method

The `__call__` method in the `GAE` class processes rewards, values from the current and next states, and done flags (indicating the end of an episode) to compute the advantages:

1. **Temporal Difference (TD) Error:** The TD error is computed for each timestep, incorporating the reward received after taking an action, the value of the next state, and the value of the current state.
2. **GAE Computation:** Starting from the end of the episode, the GAE is calculated in reverse. It updates a running sum of discounted TD errors, which represents the advantage estimate. This running sum uses both `gamma` and `lambda_` to weight the importance of immediate versus future errors.
3. **Normalization:** To ensure stable updates, the advantages are normalized, adjusting their scale to have zero mean and unit variance. This step helps manage the variance in policy gradient estimates, making training more consistent.

Experimenting with λ Values:

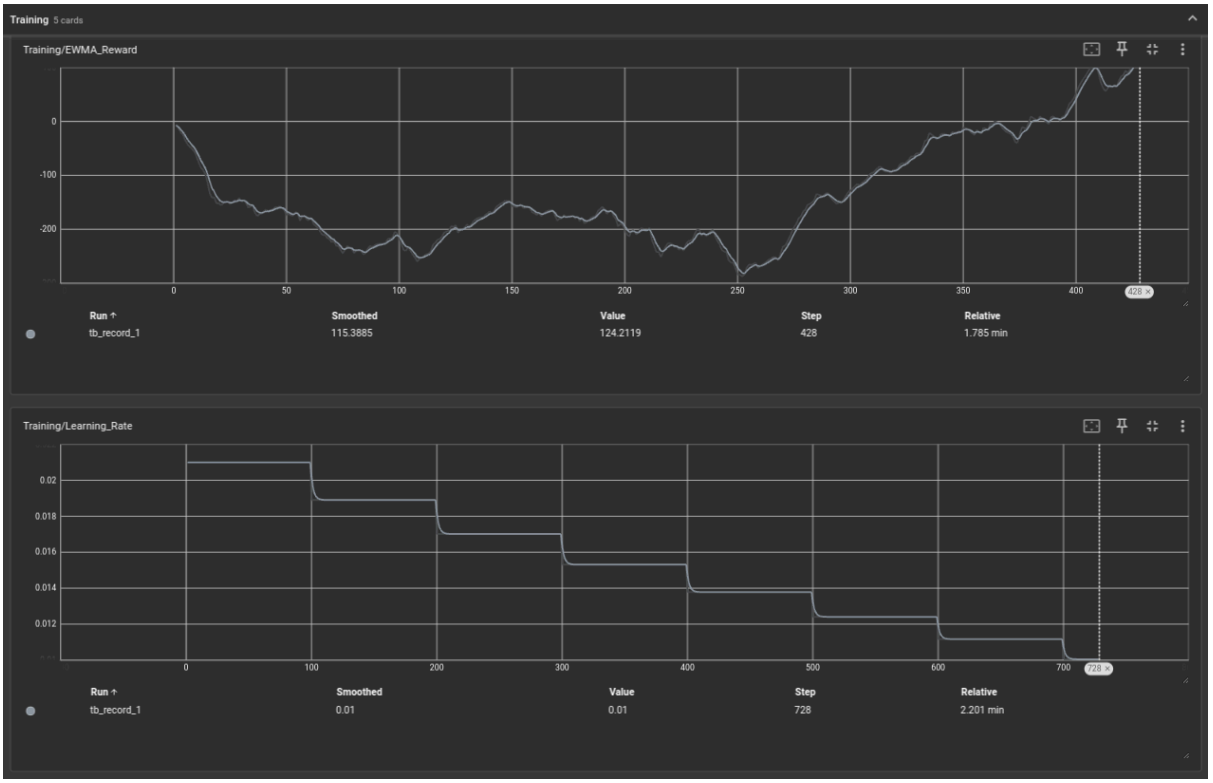
Three different λ values were tested to observe their effects on the learning dynamics and overall success of the agent:

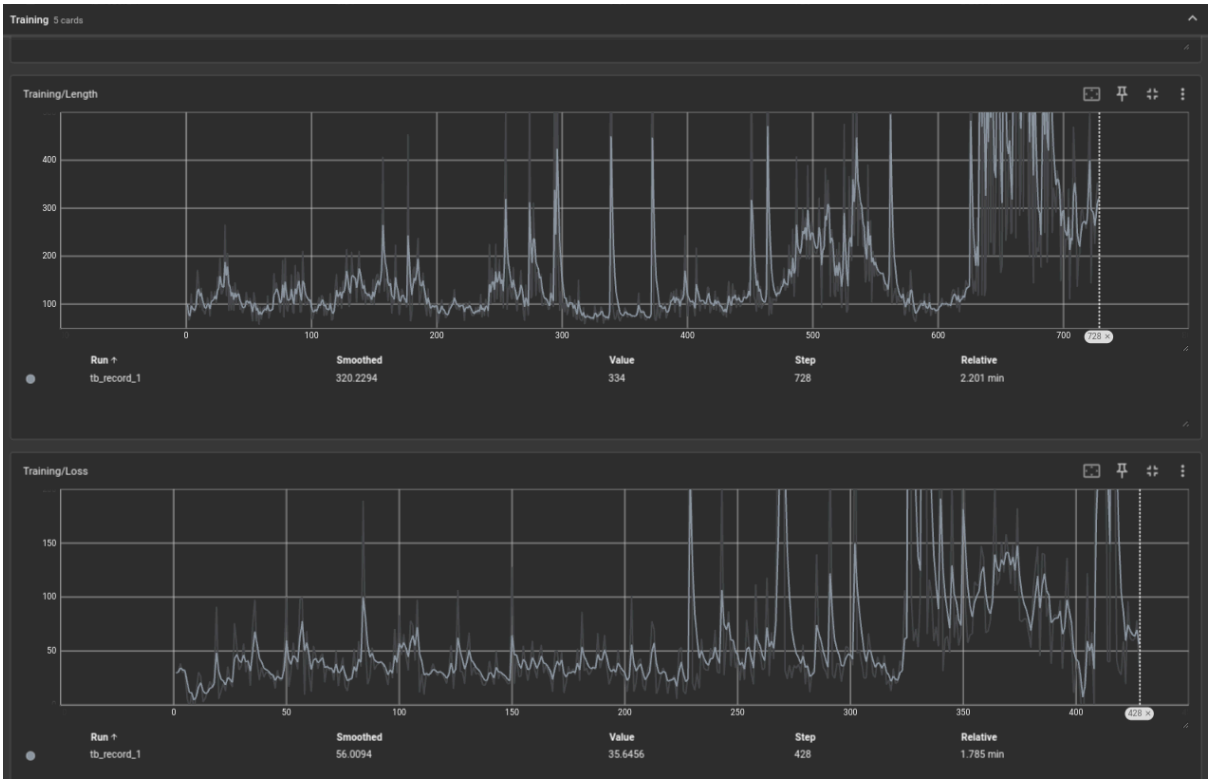
1. **λ = 0.95**: Achieved a solved status in 561 episodes, with the final episode reaching an EWMA reward of approximately 124.08. This setting provided a balance, but still left room for variance reduction.
2. **λ = 0.97**: Required 653 episodes to solve, indicating slightly less efficiency compared to λ = 0.95. The higher λ increased the weighting of future rewards, which might have introduced too much variance into the advantage estimates, thus slightly degrading the performance.
3. **λ = 0.93**: Proved to be the most effective, solving the environment in just 428 episodes. The lower λ value here likely provided a better balance by

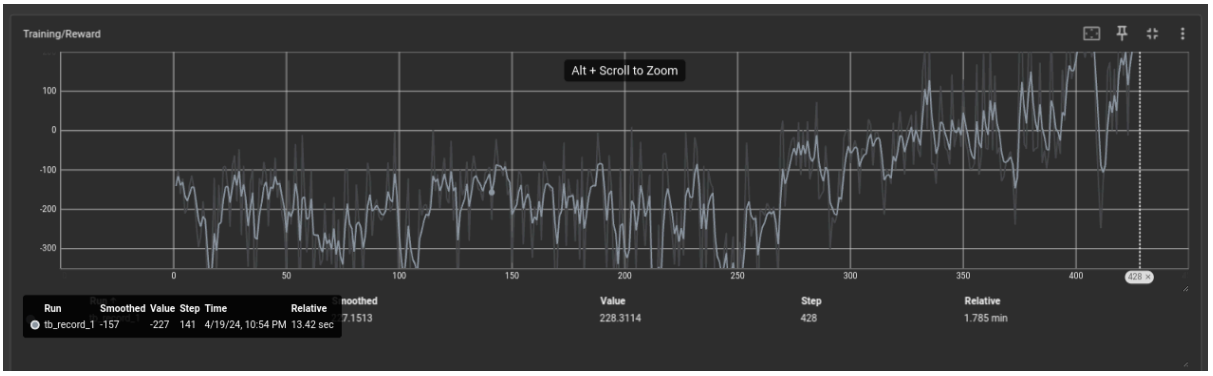
reducing variance more aggressively, which facilitated faster learning and more stable updates.

Summary of Results:

The experiments clearly showed that a lower λ can potentially improve learning speed and stability in this context, as evidenced by the number of episodes required to achieve a satisfactory reward level. λ = 0.93 not only resulted in faster convergence but also in a more robust policy as indicated by the fewer episodes required to reach the EWMA reward threshold. This suggests that for tasks like Lunar Lander, where balancing and landing dynamics can greatly vary from one state to another, a more conservative approach in estimating advantages (i.e., a lower λ) can be beneficial.




Training 5 cards
Training/Length
400
300
200
100
0
100
200
300
400
500
600
700
728
Run
Smoothed
Value
Step
Relative
tb_record_1
320.2294
334
728
2.201 min
Training/Loss
150
100
50
0
50
100
150
200
250
300
350
400
428
Run
Smoothed
Value
Step
Relative
tb_record_1
56.0094
35.6456
428
1.785 min


Training/Reward
Alt + Scroll to Zoom
100
0
-100
-200
-300
0
50
100
150
200
250
300
350
400
428
Run
Smoothed
Value
Step
Time
Relative
tb_record_1
-157
-227
141
4/19/24, 10:54 PM
13.42 sec
Value
Step
Relative
228.3114
428
1.785 min