Reinforce Learning Final Project Report

Jiabao Wu

January 17, 2026

1 Introduction

This report presents the applications of reinforcement learning techniques to the Atari games and mujoco environments. For Atari, we implemented the Dueling DQN, which is value-based. For mujoco, we implemented the PPO, which is policy-based.

2 Atari

2.1 Introduction to Dueling DQN

Dueling DQN is an improvement over the traditional DQN architecture. Different from the original DQN, it seperate the estimation of the state value function and the advantage function into two streams. As the figure below shows:

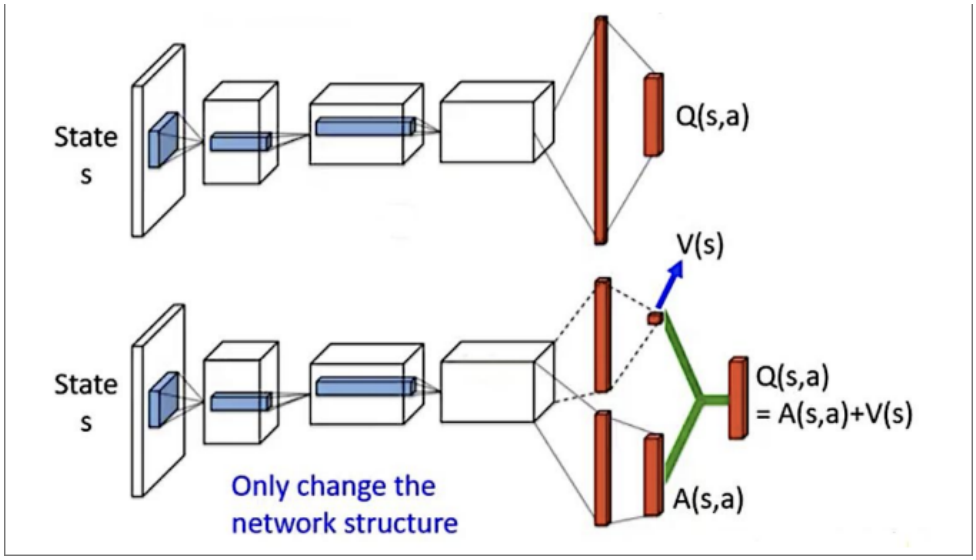


Figure 1: Structure of Dueling DQN.

After obtaining the feature value from three convolutional layers, the network splits into two streams: one for estimating the state value function $V(s)$ and another for the advantage function $A(s, a)$. We combine these two streams to get the Q value $Q(s, a)$. In order to ensure that the Q value is unique, we use the following equation to combine them:

$$Q(s,a) = V(s) + \left(A(s,a) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s,a') \right)$$

where $|\mathcal{A}|$ is the number of possible actions.

2.2 Implementation of Dueling DQN

- **Wrappers:** Since the original Atari environment has high-dimensional input and complex dynamics, we use several wrappers to preprocess the environment:

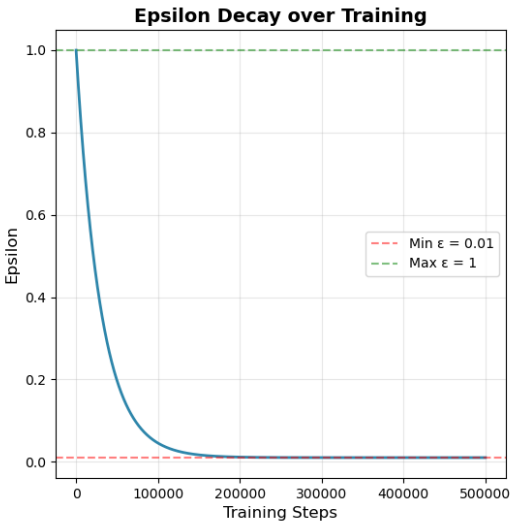
1. **AFK Reset:** Each time the environment is reset, the agent will be AFK(away from keyboard) for a random number of steps (between 1 and 20), introducing randomness in the starting state.
2. **Fire Start:** Some games need to "FIRE" to start. The wrapper will ensure that "FIRE" is taken at the beginning of each episode.
3. **One Life Reset:** For a game with multiple lives, the wrapper will set `done=True` when the agent loses a life, making each life an independent episode.
4. **Life Loss:** Each time the agent loses a life, make reward -1 and continue.
5. **Frame Stack:** Stack the last four frames to form the input state, providing temporal context to the agent. The shape of a single frame is $(84, 84, 1)$, and after stacking four frames, the input shape becomes $(84, 84, 4)$.
6. **Gray Scale and Resize:** Convert the RGB frames to grayscale and resize them to 84x84 pixels to reduce computational complexity.
7. **Reward Clipping:** Clip the rewards to the range [-1, 1] to stabilize training.

- **Epsilon-Greedy:**

We implemented an exponential decay for ϵ. The specific formula is:

$$\epsilon = \epsilon_{min} + (\epsilon_{max} - \epsilon_{min}) \cdot \exp(-i/30000)$$

where i is the frame index, $\epsilon_{max} = 1$ and $\epsilon_{min} = 0.01$.



2.3 Results on Atari

Under the hyperparameters listed below, we trained a Dueling DQN agent.

- Max Frame: 1million for **Pong-v5**, 500k for **Breakout-v5** and **Boxing-v5**
- Learning Rate: $1e-3$
- Discount Factor γ: 0.99
- Replay Buffer Capacity: 10000
- Batch Size: 32
- Target Network Update Frequency: 1000 steps
- Initial Exploration Steps: 10000 steps

On "Pong-v5", the training curve is shown below:

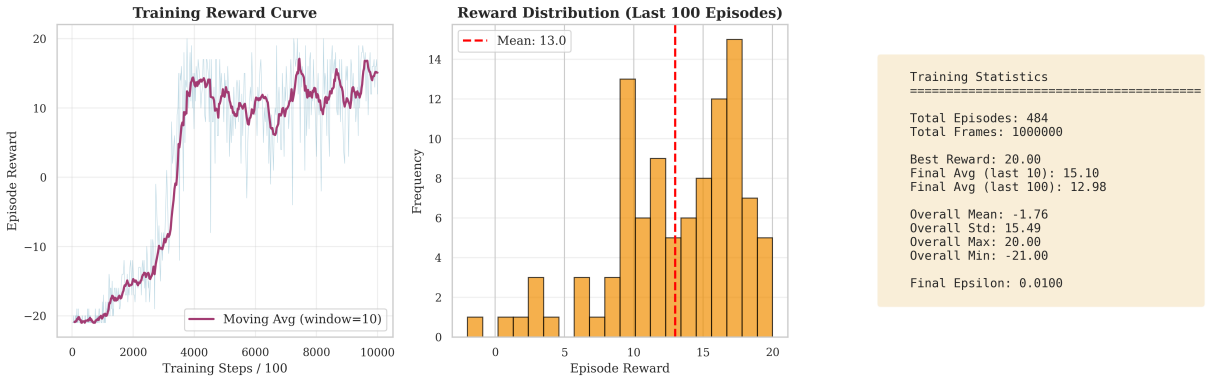


Figure 2: Training Curve on Pong-v5.

As illustrated in the figure, the reward experiences a significant surge between 300k and 400k frames, eventually trending toward convergence. The final reward stabilizes at approximately 13. We also tested the trained agent, the recorded video can be found in `atari/result/Pong-v5`.

On "Breakout-v5", the training curve is shown below:

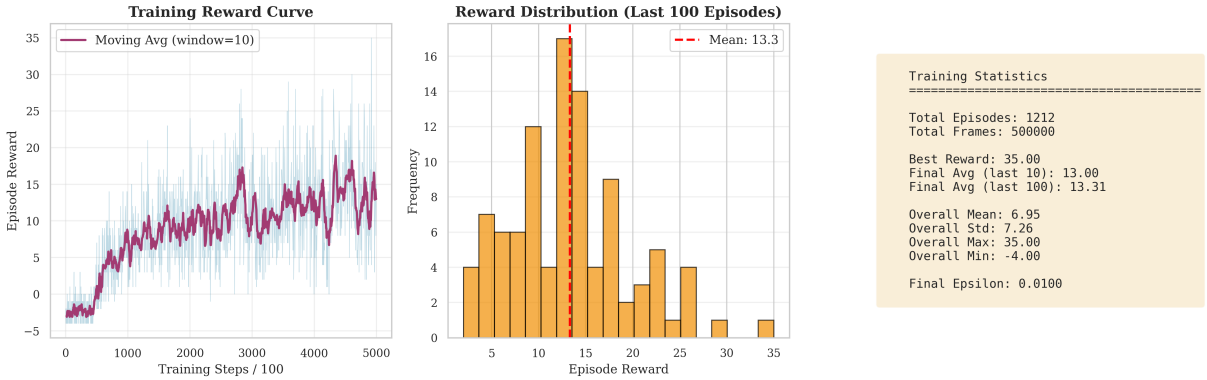


Figure 3: Training Curve on Breakout-v5.

Compared with Pong-v5, the environment of Breakout-v5 is significantly more complex. Nevertheless, the training curve maintains a steady upward trend and demonstrates clear convergence. After 500k frames of training, the agent achieves a reward of approximately 13. The recorded video can be found in `atari/result/Breakout-v5`.

On "Boxing-v5", the training curve is shown below:

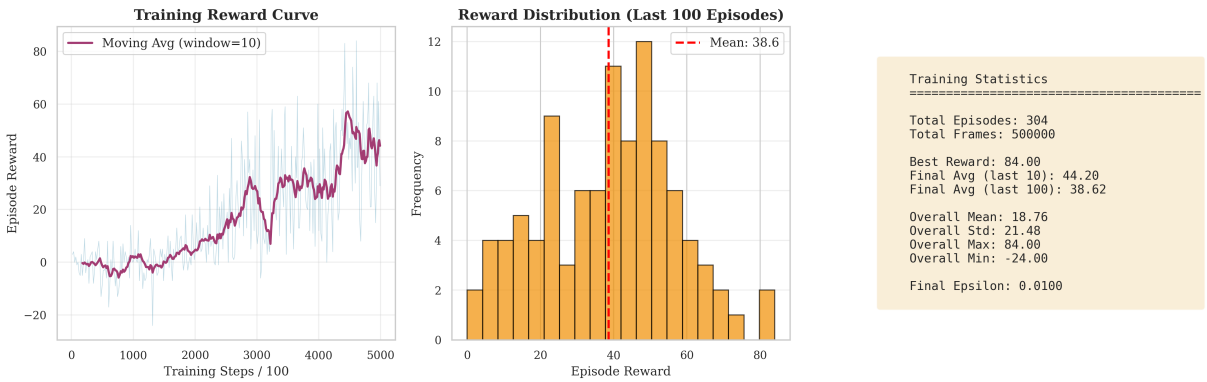


Figure 4: Training Curve on Boxing-v5.

Since Boxing-v5 has a relatively complex environment, the curve shows a slow but steady upward trend. After 500k frames, the agent achieves a reward of about 40. Video is also recorded in `atari/result/Boxing-v5`.

Here is the table of rewards of top-10 tests on three environments:

Table 1: Rewards of top-10 tests on Atari

Index	1	2	3	4	5	6	7	8	9	10	**Average**
Pong-v5	17.0	15.0	11.0	16.0	12.0	13.0	17.0	15.0	11.0	19.0	**14.6**
Breakout-v5	18.0	16.0	26.0	21.0	9.0	14.0	8.0	35.0	14.0	35.0	**19.6**
Boxing-v5	54.0	72.0	73.0	31.0	28.0	47.0	32.0	39.0	38.0	68.0	**48.2**

2.4 Ablation Studies

2.4.1 Learning Rate

As far as we know, low learning rate leads to slow convergence. What if it is too high? To investigate the impact of high learning rate, we set it to $1e-2$ and train on Breakout-v5.

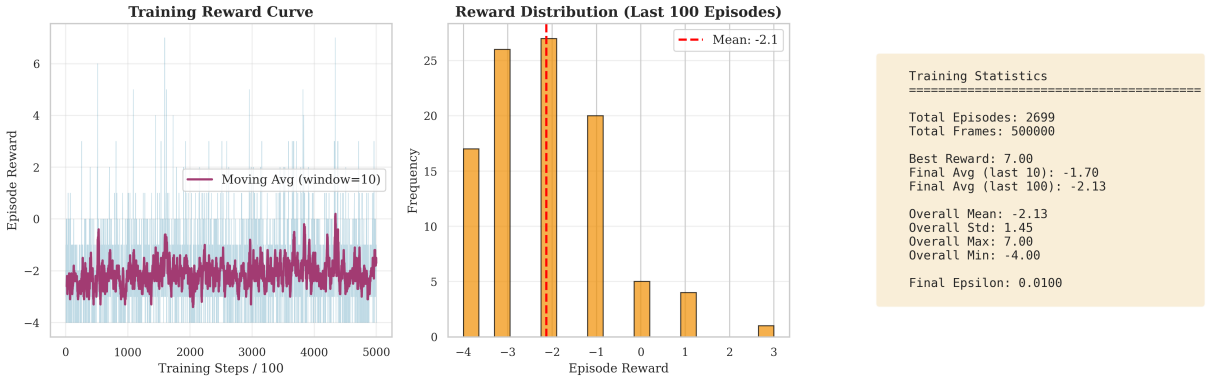


Figure 5: Training Curve of Breakout-v5 with $lr = 1e-2$.

As the figure shows, the training curve is nearly a flat line, indicating that the model with too-high learning rate does not learn anything. This is because a high learning rate leads to too large parameter updates, resulting in divergence during training.

In fact, compared with $2e-4$ and $1e-2$, setting the learning rate to $1e-3$ is of great benefit to the training. It balance the speed and stability of convergence well.

2.4.2 Epsilon

In the origin implementation, we set the minimum epsilon to 0.01. What if we always keep some spirit of exploration? We set the minimum epsilon to 0.3 and here is the result training curve on Breakout-v5.

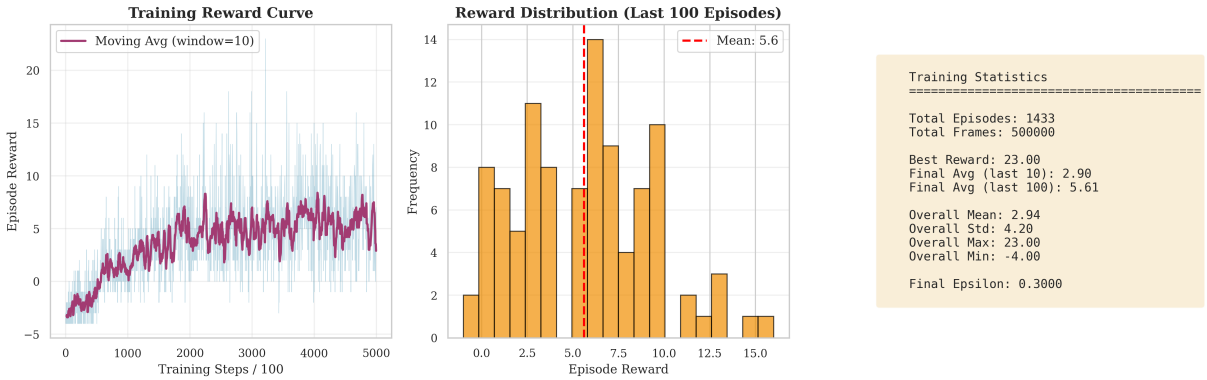


Figure 6: Training Curve of Breakout-v5 with $\epsilon_{min} = 0.3$.

The figure shows that the curve is lower than the original one and converges to a pool reward around 5. This is because a high minimum epsilon introduces too much randomness in action selection, so even in late training, the agent still takes random actions 30% of the time, wasting opportunities to exploit learned knowledge.

2.4.3 Capacity

The capacity of memory buffer influences the diversity of experience. We set it to 100k and train on Breakout-v5.

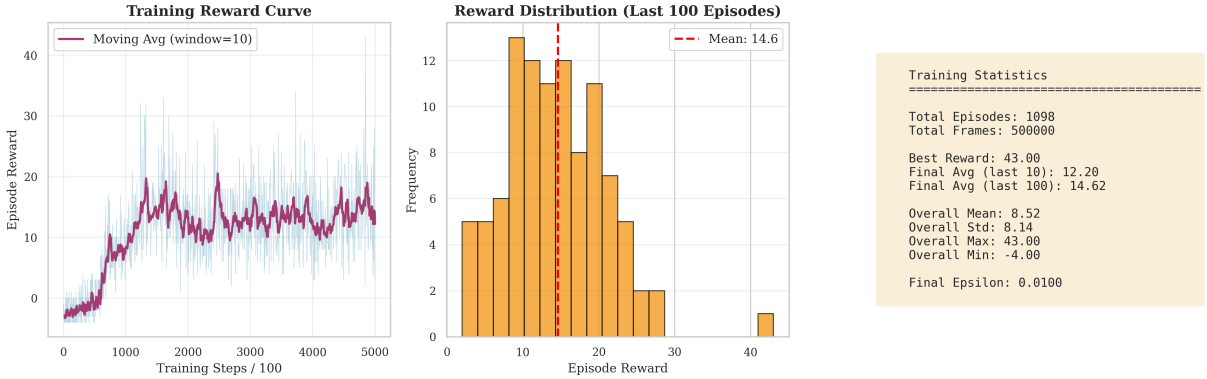


Figure 7: Training Curve of Breakout-v5 with capacity=100k.

Different from the curves above, this one is similar to the origin curve, with a steady upward trend and clear convergence. Moreover, the final reward is little higher than the original one, reaching about 15. This is because a larget capacity provides more diverse experiences for training, improving the generalization ability of the model. This explains why there is a high reward in the historigam.

2.5 Analysis of Improvement

Compared with the original DQN, Dueling DQN greatly improves the performance on Atari games. We also implemented DQN to train on Breakout-v5 with the same hyperparameters. The training curve is shown below:

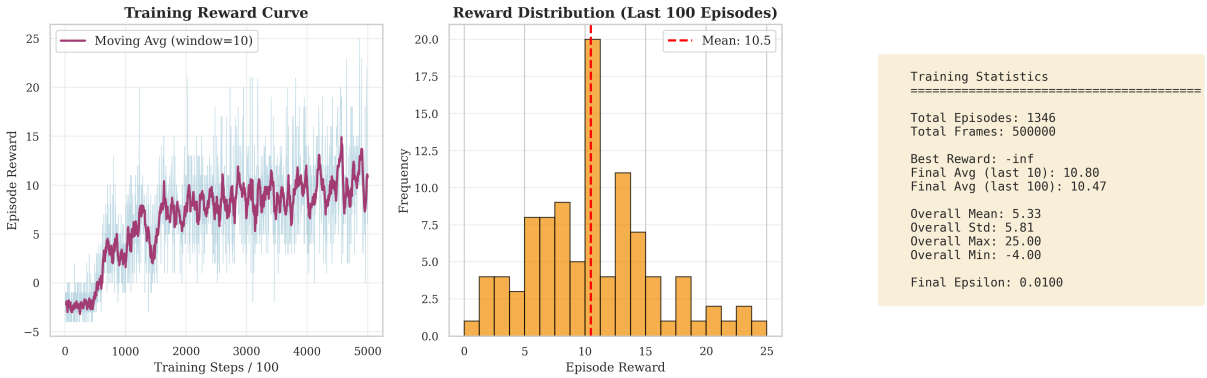


Figure 8: Training Curve of DQN on Breakout-v5.

Compared with the Figure 3, the Dueling DQN outperforms DQN. The average reward of DQN is about 10, while Dueling DQN achieves about 13. The reward curve of DQN is also lower, with more volatility, than Dueling DQN.

In fact, the superiority of Dueling DQN in Breakout-v5 can be attributed to its ability to learn the state-value function $V(s)$ independently. In Atari environments, the exact action taken may not matter in states where the ball is still far from the paddle. Dueling DQN captures this by estimating the value of being in such a 'safe' state, leading to faster and more robust learning compared with DQN.

2.6 Conclusion

In this section, we successfully implemented and evaluated Dueling DQN on Atari environments. Experimental results demonstrate that the agent achieves solid performance even within a limited training frames. Our ablation studies further reveal that learning rate, exploration rate (ϵ), and replay buffer capacity are critical factors influencing training stability and final rewards.

Furthermore, a direct comparison with original DQN highlights the superiority of the dueling architecture. By decoupling state-value $V(s)$ and advantage $A(s, a)$, the model exhibits smoother convergence and higher sample efficiency, confirming that the dual-stream approach is more effective for complex discrete-action Atari tasks.

3 Mujoco

3.1 Introduction to PPO

Proximal Policy Optimization (PPO) is a prominent policy-based reinforcement learning algorithm designed to improve the stability and reliability of policy updates. Unlike standard Policy Gradient methods that are sensitive to step sizes, PPO introduces a clipped surrogate objective to prevent excessively large policy updates. The core architecture often follows an Actor-Critic framework, as shown in the figure below:

PPO maintains two versions of the policy: the current policy π_θ and the old policy $\pi_{\theta_{old}}$ from which the data was sampled. The update is guided by the probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$. To ensure stable convergence, PPO optimizes the following clipped objective function:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

where $\hat{A}_t$ is the estimated advantage at time t, and ϵ is a hyperparameter (typically 0.1 or 0.2) that limits the change of the policy in a single update. This mechanism ensures that the agent learns conservatively, avoiding catastrophic performance collapses.

3.2 Implementation of PPO

To implement PPO, we firstly construct the Actor-Critic network architecture. The actor network outputs the mean and standard deviation of a Gaussian distribution for continuous action spaces, while the critic network estimates the state value function.

- **Normalization:** We normalize the observations to have zero mean and unit variance, which helps stabilize training.
- **Advantage Estimation:** We use Generalized Advantage Estimation (GAE) to compute the advantage function, which balances bias and variance in the estimates.

3.3 Results on Mujoco

Under the hyperparameters listed below, we trained a PPO agent.

Table 2: Hyperparameters for PPO on Mujoco

Hyperparameter	**Hopper-v4**	**HalfCheetah-v4**	**Ant-v4**
Max Episodes	3000	2000	2000
Learning Rate	$1e-3$	$3e-4$	$3e-4$
Max Steps γ	2048	2048	2048
GAE Parameter λ	0.98	0.98	0.98
Clip Range ϵ	0.1	0.1	0.1
Batch Size	64	64	64
Update Epochs	10	10	10
Entropy Coefficient	0.01	0.001	0.001

On "Hopper-v4", the training curve is shown below:

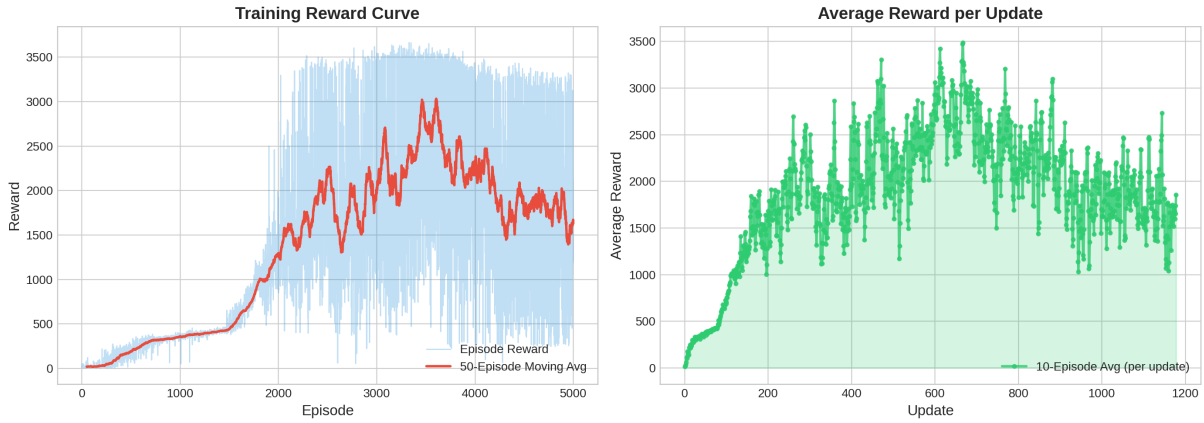


Figure 9: Training Curve on Hopper-v4.

As we can clearly see from the graph, the reward shows a rapid increase in the initial training phase, followed by a gradual convergence. The final reward stabilizes at around 3000. Test videos can be found in `mujoco/result/Hopper-v4`.

On "HalfCheetah-v4", the training curve is shown below:

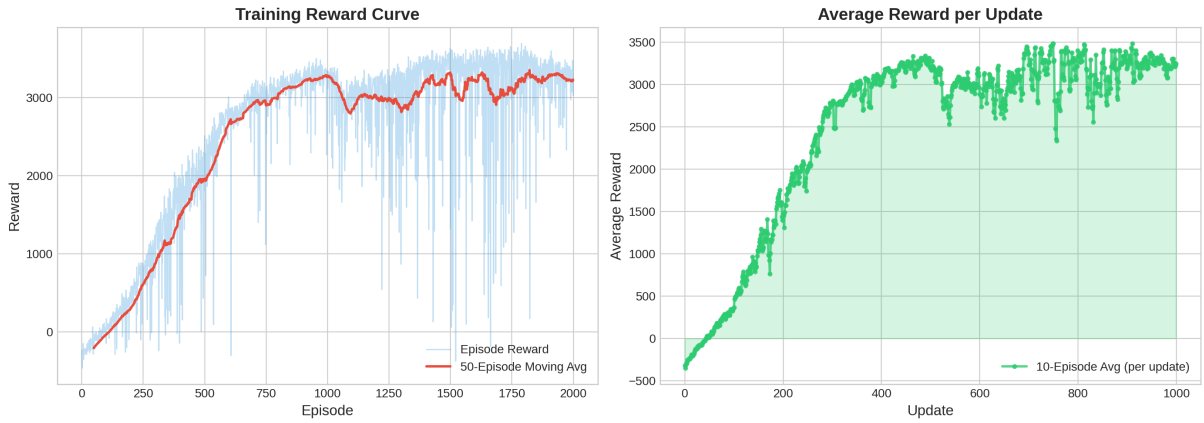


Figure 10: Training Curve on HalfCheetah-v4.

Since the environment is more complex, we set the learning rate to $3e-4$ to ensure stabler training and set the entropy coefficient to 0.001 to disencourage over-exploration. Test videos can be found in `mujoco/result/HalfCheetah-v4`.

On "Ant-v4", the training curve is shown below:

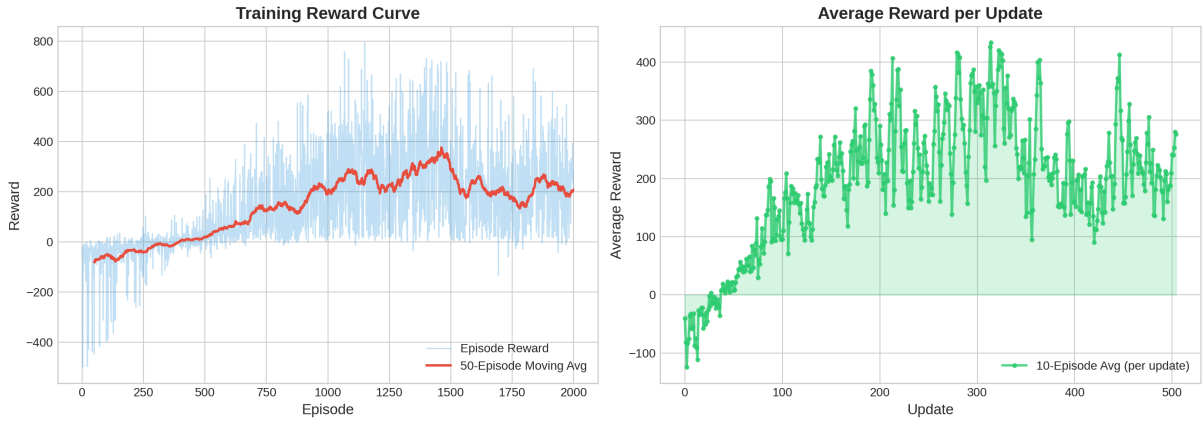


Figure 11: Training Curve on Ant-v4.

The environment is the most complex among the three. The curve is also relatively volatile, but it still shows an upward trend. The curve seems to converge to about 800 after 2000 episodes. Test videos can be found in `mujoco/result/Ant-v4`.

The table below shows the rewards of top-5 tests on three environments:

Table 3: Rewards of top-5 tests on Mujoco

Index	1	2	3	4	5	**Average**
Hopper-v4	3252.00	3265.31	3254.47	3259.20	3252.22	**3256.64**
HalfCheetah-v4	3668.66	3611.04	3614.78	3540.17	3634.25	**3613.78**
Ant-v4	856.25	675.16	1056.97	1007.50	1135.20	**946.21**

3.4 Ablation Study

In this section, we try to analyse the impact of learning rate and entropy coefficient on the training performance of PPO. Since Hopper-v4 is relatively a simple environment, we conduct experiments on it by varying these hyperparameters while keeping others constant.

3.4.1 Learning Rate

This time we set lr to $1e-2$ and train.

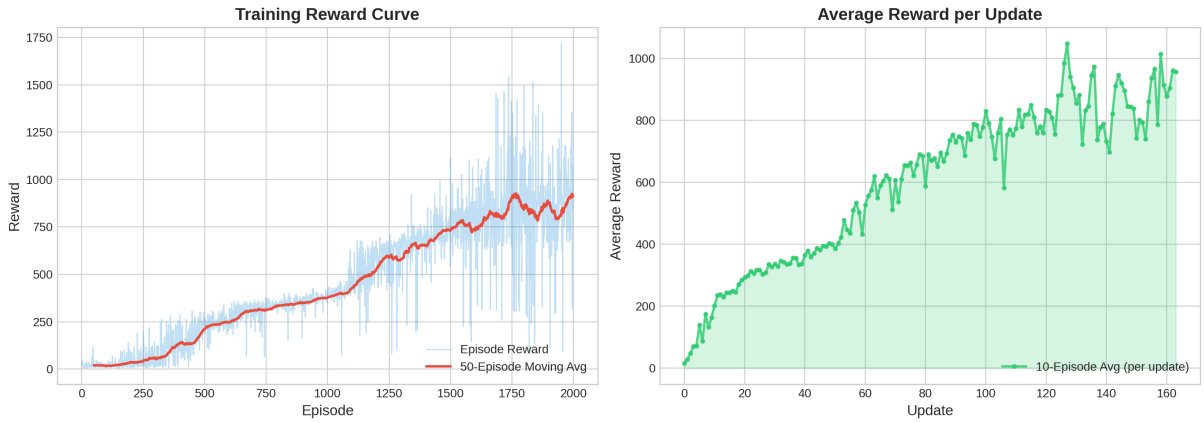


Figure 12: Training Curve of Hopper-v4 with $lr = 1e-2$.

To our suprise, this curve shows a rapid increase in reward, reaching about 1000 within 2000 episodes. However, after that, the reward has high variance and does not reach the expected reward fianlly. This indicates that too high learning rate may lead to instablity during training, causing unexpected performance drops.

3.4.2 Entropy Coefficient

In PPO, the entropy coefficient controls the exploration-exploitation trade-off. A higher entropy coefficient encourages more exploration by penalizing certainty in action selection. We set it to 0.5 and train on Hopper-v4.

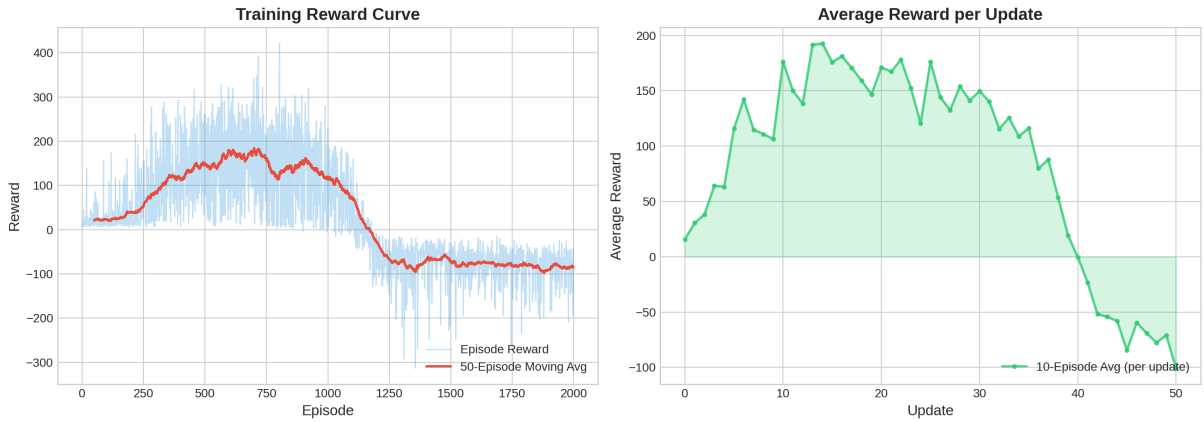


Figure 13: Training Curve of Hopper-v4 with $ent\_coef = 0.5$.

The result, in our expectation, shows that the training is failed. The reward climbs up at the beginning, but soon drops to negative and stays there. This is because a high entropy coefficient leads to excessive exploration, preventing the agent from exploiting learned policies effectively.

3.5 Conclusion

In this section, we successfully implemented and evaluated PPO on Mujoco environments. Experimental results demonstrate that the agent achieves solid performance even within a limited training episodes. Our ablation studies further reveal that learning rate and clip range are critical factors influencing training stability and final rewards.

Due to limited time, we did not conduct a comprehensive ablation study on all hyperparameters. The implemented network of PPO still requires futher tuning to avoid high variance during training.