NYCU_Artificial_Intelligence Capstone_HW2

- This pdf is available at: here (https://hackmd.io/@Origamyee/HyIz7Oyyle)
- This code is available at: here (https://github.com/hsiuyee/NYCU_Artificial_Intelligence_Capstone/blob/main/Lab2/AI_HW2.ipynb)

Part 1. Overview

In this project, we explore how different reward mechanisms and algorithmic hyperparameters affect the learning performance of reinforcement learning agents trained with Proximal Policy Optimization (PPO).

We focus on five key experiments:

1. **Dense vs. Sparse Reward**
   We compare standard dense rewards with a sparse reward version that only gives feedback at the end of each episode. This allows us to understand the challenge of exploration when the reward signal is limited.
2. **Sparse vs. Sparse + Intrinsic Reward**
   Inspired by curiosity-driven learning, we test whether adding an intrinsic reward (based on state novelty) can improve performance in sparse environments.
3. **Scaling Intrinsic Reward**
   We investigate the effect of changing the strength (scaling factor α) of intrinsic rewards on agent performance. While strong curiosity can boost exploration, it may also lead to instability.
4. **Clipping Threshold in PPO**
   We analyze how different values of the `clip_range` parameter in PPO affect learning speed and stability. We specifically look for signs of overshooting and undershooting in policy updates.
5. **Algorithm Comparison: PPO vs. A2C on Atari**
   To include an Atari benchmark and compare different policy-gradient architectures, we train both PPO and A2C agents on **Assault-v5**. This experiment highlights how PPO's clipped surrogate objective improves stability and sample efficiency compared to the vanilla Advantage Actor-Critic (A2C) in a high-dimensional, delayed-reward environment.

All experiments are conducted in the `CartPole-v1` environment—except Experiment 5, which uses `Assault-v5` —and results are evaluated using **dense rewards** to reflect each agent's true capabilities in the original task. Video demonstrations and learning curves are provided for visual inspection.

Part 2. Experiments

Experiment 1 - Dense vs. Sparse reward

> What if we give the agent a sparse reward instead of a dense reward ?

- **Environments / Tasks:**
  - We use the classic control task **CartPole-v1** from Gymnasium, comparing two reward schemes:
    1. **Dense reward** (original environment reward)
    2. **Sparse reward** (intermediate reward = 0; terminal reward = total_reward / 2)
- **Algorithm & Training:**
  - Agents are implemented with **PPO** (`MlpPolicy`) from Stable-Baselines3.
  - Each model is trained for **30 epochs**, with **1 000 timesteps per epoch** (`total_timesteps=1000` , `reset_num_timesteps=False`).
  - We wrap the environment in a `Monitor` to log per-episode returns, run **20 evaluation episodes** (using the same reward wrapper) after each epoch, and record videos for qualitative comparison.

We design a sparse reward mechanism as follows:

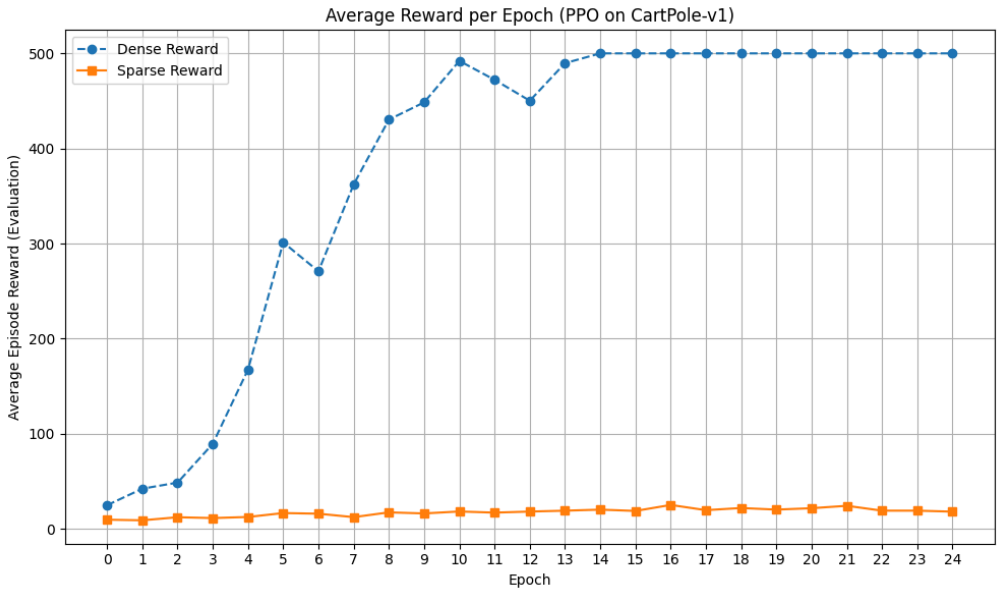
1. At each intermediate step, we give a reward of 0 (hiding the original reward).
2. Upon termination or truncation, we assign a reward equal to half of the accumulated hidden reward.

Intuitively, when we use sparse reward mechanism to train our PPO model, it performs poorly because it receives no rewards or guidance during exploration.

As we can see, the performance difference between the two reward mechanisms is significant. When using dense rewards, we observe approximately a 10x improvement. The results are shown below.

Dense reward	Sparse reward
Video (https://www.youtube.com/watch?v=txS6QbuBoLI&ab_channel=HsiuyeeLiao%28Origami%29)	Video (https://www.youtube.com/watch?v=DN0-_K-j6nY)

Reward Comparison:



Experiment 2 – Sparse vs. Sparse + Intrinsic Reward

> What if we add intrinsic reward to improve training in a sparse environment?

- **Environments / Tasks:**
  - **CartPole-v1** with two custom wrappers:
    1. **Sparse Reward:** intermediate = 0; terminal = total_reward / 2
    2. **Sparse + Intrinsic Reward:** same sparse scheme **+** count-based bonus (+1 on first visit to each rounded state)
- **Algorithm & Training:**
  - **PPO** (`MlpPolicy` , Stable-Baselines3 defaults)

- **Training loop:**
  - 30 epochs × 1 000 timesteps per epoch
    (`model.learn(total_timesteps=1000, reset_num_timesteps=False)`)
  - After each epoch, run 20 evaluation episodes (wrapped env) via `Monitor` to log returns

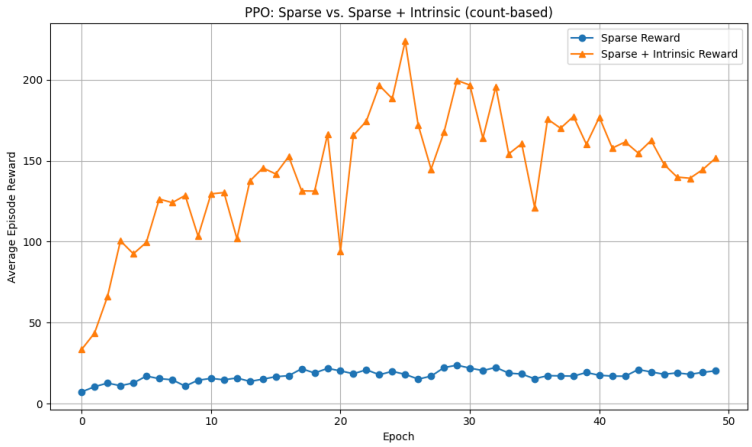
This experiment idea comes from the paper Curiosity-driven Exploration by Self-supervised Prediction (https://proceedings.mlr.press/v70/pathak17a/pathak17a.pdf), which focuses on the problem of learning in sparse reward environments. The main idea is that, since extrinsic rewards are very sparse, we need to introduce intrinsic rewards to motivate the model to learn. Therefore, we design a simple intrinsic reward mechanism as follows.

We adopt the same idea from the above paper. We want the model to maintain its curiosity during exploration. Therefore, we add intrinsic rewards when the model visits previously unvisited places.

As we can see, the performance difference between the two reward mechanisms is significant. When using intrinsic rewards, we observe approximately a 2x improvement. The results are shown below.

Sparse reward	**Sparse + Intrinsic reward**
Video (https://www.youtube.com/watch?v=laO5cCuUD9k&ab_channel=HsiuyeeLiao%28Origami%29)	Video (https://www.youtube.com/watch?v=rZ6Ooru7_oE&ab_channel=HsiuyeeLia

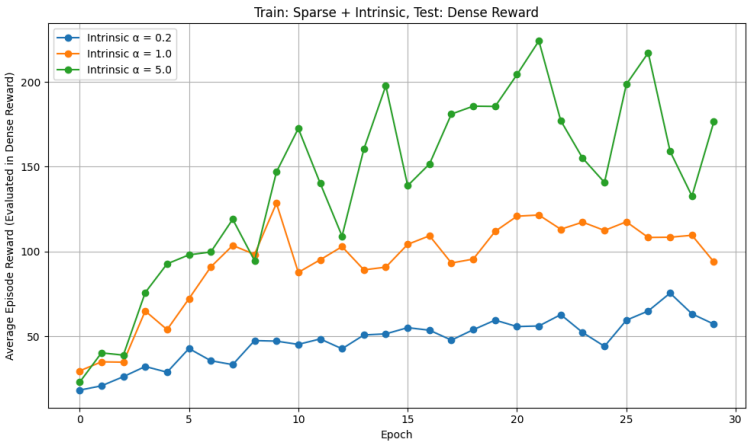
Reward Comparison:



Experiment 3 – Scaling Intrinsic Reward

> What if we scale intrinsic rewards?

- **Environments / Tasks:**
  - **CartPole-v1** with **Sparse + Intrinsic Reward** wrapper:
    1. Sparse scheme: intermediate = 0; terminal = total_reward / 2
    2. Intrinsic bonus: +α on first visit to each rounded state
    3. **α values tested:** 0.2, 1.0, 5.0
- **Algorithm & Training:**
  - **PPO** (`MlpPolicy` , Stable-Baselines3 defaults)
  - **Training loop:**
    - 30 epochs × 1 000 timesteps per epoch (`model.learn(total_timesteps=1000, reset_num_timesteps=False)`)
    - After each epoch, evaluate **20 episodes** in the **dense** CartPole-v1 environment via `Monitor`



alpha = 0.2	Clip = 1.0
Video (https://www.youtube.com/watch?v=HRdEqnmBs8w&ab_channel=HsiuyeeLiao%28Origami%29)	Video (https://www.youtube.com/watch?v=UYvrO9dsDiE&ab_channel=Hsiuyee

As shown in the figure above, we trained the agent using only Sparse + Intrinsic Reward, but evaluated its performance using Dense Reward after each epoch to reflect its actual capability in the original environment.

- When $\alpha = 0.2$, the agent struggles to learn effectively — the dense reward remains below 60 throughout training.
- With $\alpha = 1.0$, the agent performs significantly better, reaching average dense rewards around 130–160, indicating that moderate intrinsic rewards can effectively guide useful exploration.
- Interestingly, $\alpha = 5.0$ achieves the best overall performance, with dense rewards often above 200 and occasionally exceeding 250. This suggests that in sparse environments, strong exploration incentives can lead to substantially better policies — particularly during early to mid training.

However, the high-α setting also exhibits more performance fluctuation, possibly due to excessive curiosity leading to over-exploration and reduced stability.

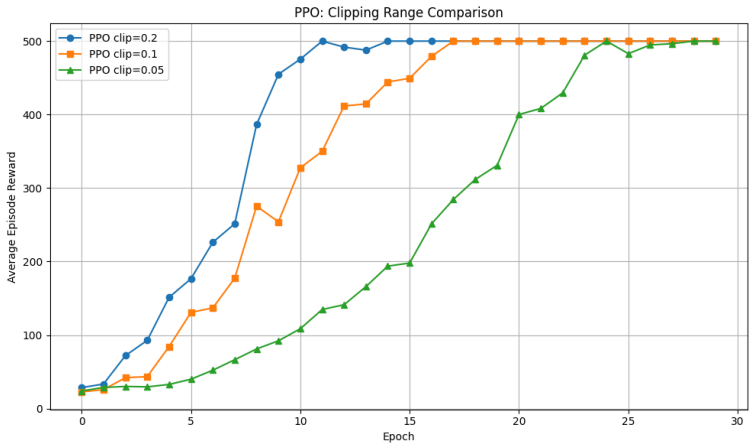
In short, scaling intrinsic rewards matters — and in this experiment, larger α values yield better test-time performance, as long as the exploration remains sufficiently focused.

Experiment 4 – PPO clip

> What if we change the clipping threshold parameter ?

- **Environments / Tasks:**
  - **CartPole-v1** (no wrapper)
- **Algorithm & Training:**
  - **PPO** (`MlpPolicy` , Stable-Baselines3 defaults)
  - **Hyperparameter tested:** `clip_range` ∈ {0.2, 0.1, 0.05}
  - **Training loop:**
    - 30 epochs × 1 000 timesteps per epoch (`model.learn(total_timesteps=1000, reset_num_timesteps=False)`)
    - After each epoch, evaluate **20 episodes** in the same env via `Monitor`

This experiment is designed to investigate how the clipping range parameter (clip_range) in PPO affects learning dynamics — particularly in terms of overshooting (over-aggressive updates) and undershooting (overly conservative updates).



Clip = 0.1	Clip = 0.2	Clip = 0.05
Video (https://www.youtube.com/watch?v=NRo_xhy3uPA)	Video (https://www.youtube.com/watch?v=gTlv1Xg22XA)	Video (https://www.youtube.com/watch?v=xUH_yZmG1Fk)

As shown in the figure and video above:

- When clip = 0.2, the agent learns very quickly, reaching the maximum reward in just around 10 epochs. However, we observe larger fluctuations in performance after convergence. This suggests that a large clipping range allows aggressive policy updates, but may cause overshooting and instability.
- With clip = 0.1, the agent still learns efficiently and converges smoothly. The oscillation is smaller than with clip = 0.2 and clip = 0.05, indicating this value offers a balanced trade-off between learning speed and stability.
- In contrast, clip = 0.05 leads to very slow learning in the early stages. Although the agent eventually performs well, the updates are likely too cautious — a classic sign of undershooting.

That said, the differences between overshooting and undershooting are not extreme in this experiment, mainly because we are using a relatively simple environment: CartPole-v1. In more complex environments, such as those with

high-dimensional state spaces or delayed rewards, improper clipping could have a much more drastic effect on performance.

In short:

- Larger clip → faster updates, higher risk of overshooting
- Smaller clip → safer updates, risk of undershooting
- clip = 0.1 appears to give the most stable and effective learning under our current setting.

Experiment 5 – Atari: PPO vs. A2C

> What if we compare different policy-gradient algorithms in an Atari environment?

- **Environments / Tasks:**
  - **Assault-v5** (Gymnasium Atari via `ALE/Assault-v5`), evaluated with the original dense reward.
- **Algorithm & Training:**
  - **PPO** and **A2C** (`CnnPolicy` , Stable-Baselines3 defaults)
  - **Training loop:**
    - Total **400 000 timesteps**, broken into **80 epochs** of **5 000 timesteps** each
    - After each 5 000-step chunk, run **5 evaluation episodes** via `Monitor` to log returns
    - Progress shown with a `tqdm` bar

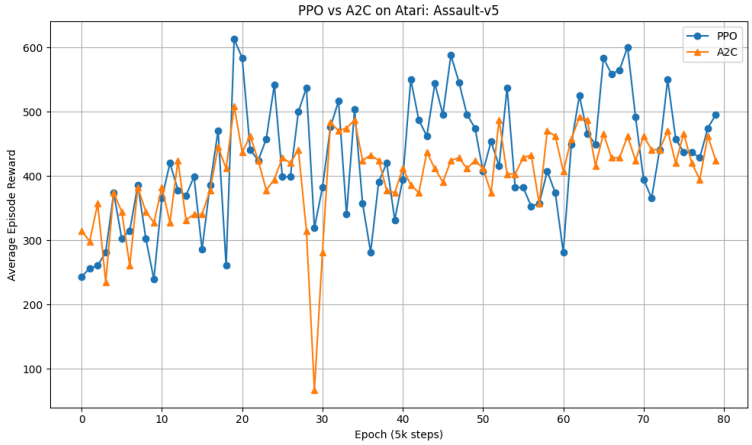
Why PPO vs. A2C?

- **Same family, fair comparison.** Both PPO and A2C are policy-gradient (actor-critic) methods, so this isolates the effect of PPO's improvements rather than comparing entirely different classes of algorithms.
- **Baseline vs. improved design.** A2C provides a simple, vanilla on-policy baseline. PPO builds on A2C/A3C by adding a **clipped surrogate objective**

(trust-region-like update) and supports larger mini-batches, which in theory should yield more stable, sample-efficient learning.

- **Stability in high-dimensional tasks.** Atari games like Assault have complex visuals and delayed rewards. We expect A2C's unconstrained updates to suffer more from instability or collapse, whereas PPO's clipping should guard against large policy swings.

Learning Curves:



- **PPO** converges faster and to higher scores.
- **A2C** learns more slowly and has pronounced drops (e.g. around epoch 30).

Video Demonstrations:

PPO (Assault-v5)	A2C (Assault-v5)
Video (https://www.youtube.com/embed/3oUSOMB_WsU)	Video (https://www.youtube.com/embed/YyOXOOLver

In short, this comparison confirms that PPO's clipped objective delivers **greater stability** and **better sample efficiency** in challenging, high-dimensional environments—exactly the benefits we hypothesized when choosing these two closely related algorithms.

Part 3. Discussion

Through five experiments, we observed:

- **Exp 1: Dense vs. Sparse Reward**
  Sparse feedback severely hampers learning (≈10× slower), whereas dense rewards provide continuous guidance and much faster improvement.

- **Exp 2: Sparse vs. Sparse + Intrinsic Reward**
  A simple count-based intrinsic bonus restores exploration in the sparse setting, roughly doubling performance compared to pure sparse rewards.
- **Exp 3: Scaling Intrinsic Reward**
  The intrinsic reward scale α trades off speed and stability:
  - $\alpha = 0.2$ under-motivates exploration (low dense scores),
  - $\alpha = 1.0$ balances exploration/exploitation (dense rewards ≈130–160),
  - $\alpha = 5.0$ maximizes peak performance (>200) but increases volatility.
- **Exp 4: PPO Clipping Threshold**
  Varying `clip_range` reveals overshooting (large clip = 0.2) vs. undershooting (small clip = 0.05);
  clip = 0.1 offers the best balance of learning speed and stability on CartPole-v1.
- **Exp 5: PPO vs. A2C on Atari**
  In the high-dimensional Assault-v5 task, PPO's clipped surrogate objective converges faster, reaches higher scores, and yields smoother learning than vanilla A2C, demonstrating the value of PPO's stability mechanisms.

Key Takeaway:
Careful **reward shaping**, appropriate **intrinsic motivation**, and built-in **policy safeguards** (like clipping) are essential for robust, efficient reinforcement learning—especially as tasks grow in complexity.

Future Work:

- Extend intrinsic reward and clipping studies to other Atari titles or 3D simulators.
- Investigate whether combining curiosity bonuses with advanced policy constraints further improves stability and sample efficiency.
- Explore newer on-policy and off-policy algorithms to generalize these insights across diverse environments.

Part 4. References

1. Stable-Baselines3 Documentation (https://stable-baselines3.readthedocs.io/en/master/)