AI capstone HW2

My Github for HW2 : link (https://github.com/jimmy93029/NYCU_Artificial_Intelligence_Capstone_Labs/tree/main/Lab2)

Introduction

Autonomous driving, like Tesla's self-driving technology, relies heavily on reinforcement learning (RL). This project explores how RL algorithms handle action constraints — for example, simulating a damaged car with limited control. We review key RL methods and evaluate their performance under such constraints, comparing model-free, model-based, and imitation learning approaches.

Algorithms

1. Basic RL: REINFORCE

$$\nabla_\theta V^{\pi_\theta}(\mu) \approx \mathbb{E}\left[\sum_t \gamma^t (G_t - V(s_t)) \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right]$$

REINFORCE updates the policy by reinforcing actions that lead to higher returns. Using a baseline like the value function helps reduce gradient variance and stabilize learning.

2. Model-Free: A2C

$$\nabla_\theta J(\theta) = \mathbb{E}\left[A^\pi(s,a) \nabla_\theta \log \pi_\theta(a \mid s)\right], \quad A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$

A2C improves over REINFORCE by estimating the advantage function and updating policies more efficiently, without requiring full trajectories.

3. Model-Free: PPO

$$\nabla_\theta J(\theta) \approx \mathbb{E}\left[\min\left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta),\ 1-\epsilon,\ 1+\epsilon)\ \hat{A}_t\right)\right]$$

PPO limits policy updates using a clipped objective, stabilizing training and maintaining performance. It balances exploitation and exploration effectively for control tasks.

4. Model-Free: SAC

$$J(\pi) = \mathbb{E}\left[Q(s,a) - \alpha \log \pi(a \mid s)\right]$$

SAC maximizes both reward and entropy to encourage exploration. Its off-policy and entropy-regularized nature makes it robust and efficient in continuous environments.

5. Model-Based: PETS

PETS uses an ensemble of learned dynamics models to simulate future outcomes. It plans actions by sampling trajectories and choosing those with highest predicted return, handling uncertainty via model diversity.

6. Imitation Learning: GAIL

GAIL trains a policy through adversarial learning by matching expert behavior. A discriminator provides rewards by distinguishing agent vs expert actions, enabling imitation without direct access to environment rewards.

Environment and Reference

BipedalWalker

BipedalWalker is a continuous control task where the agent uses two legs to walk across uneven terrain. It requires coordination and balance, with delayed rewards and penalties for falling. It's often used to test locomotion and control in robotics.

Assault-v5

Assault-v5 is a pixel-based Atari game where the agent controls a cannon to shoot enemies while dodging attacks. It has discrete actions and sparse rewards, making it a benchmark for vision-based and reactive policy learning.

Libraries

stable-baselines3

`stable-baselines3` is a popular Python library for model-free reinforcement learning. It provides reliable and well-tested implementations of common algorithms such as PPO, A2C, DDPG, TD3, and SAC. The library is built on

PyTorch and is widely used for benchmarking and fast experimentation in continuous and discrete environments.

mbrl

`mbrl` (Model-Based Reinforcement Learning Library) is a modular and flexible framework developed by Facebook AI for building and evaluating model-based RL algorithms. It supports methods like PETS and provides utilities for dynamics model learning, planning, and policy evaluation. It’s especially suited for research in data-efficient RL.

imitation

`imitation` is a PyTorch-based library built on top of `stable-baselines3` for imitation learning. It includes algorithms like Generative Adversarial Imitation Learning (GAIL). The library is designed for learning from expert demonstrations and evaluating how closely an agent can mimic expert behavior without access to the environment’s reward.

Task1 : REINFORCE in Atari task Assault-v5

Youtube : link1 (https://www.youtube.com/shorts/SwdbzL6CchQ), link2 (https://youtube.com/shorts/KROaTskq9N4?feature=share), link3 (https://youtube.com/shorts/gKVuWeiSxgI)

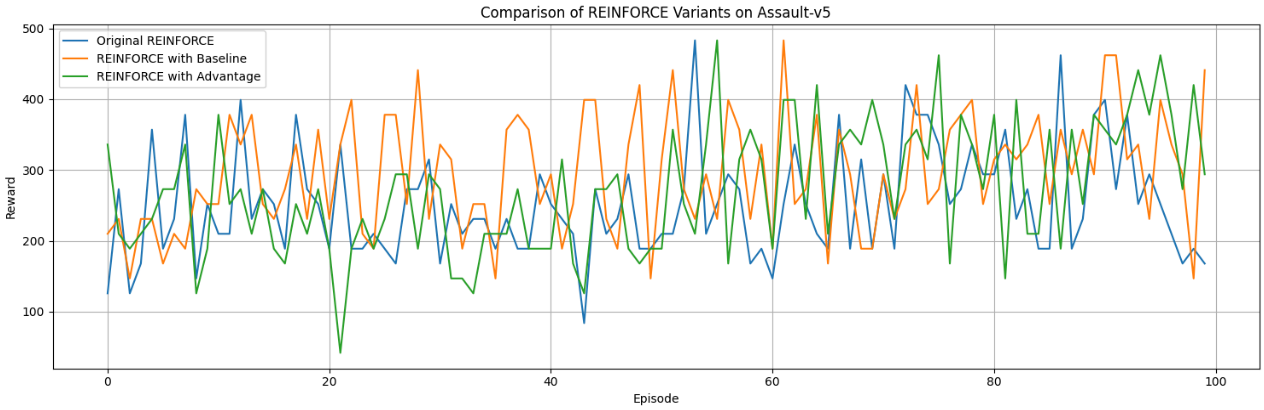
A toy example explaining the important of baseline

In REINFORCE, introducing a baseline B(s)
B(s) reduces the variance of the policy gradient estimate without introducing bias. A well-chosen baseline (e.g., the state value V(s) helps stabilize learning by centering the reward signal.

In the toy example, without a baseline, the gradient direction varies depending on the sampled action, leading to high variance. When using a baseline equal to V(s), the stochastic gradients become symmetric, and the expected update aligns with the true policy gradient, improving learning stability and efficiency.

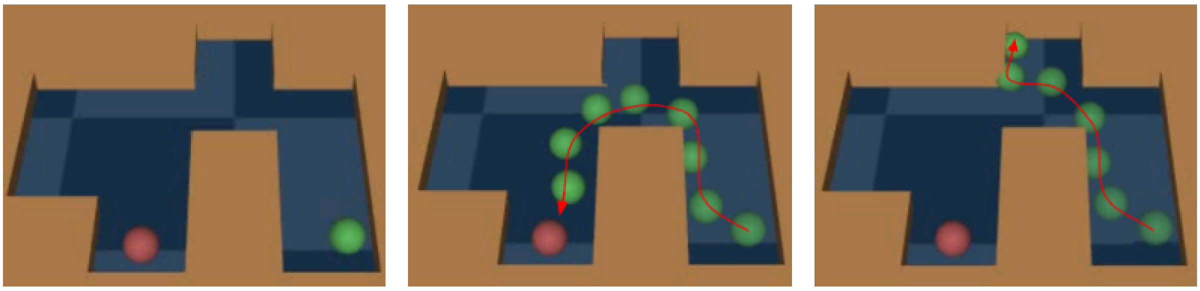
experiments

the picture is the training curve of the three algorithms. In



algorithm	REINFORCE	REINFORCE - baseline mean	REINFORCE - baseline value function
rewards	420.0	436.8	413.7

What is Action constraint



(a) Starting point (b) Unconstrained case (c) Action-constrained case

Figure 1: (a) The green sphere starts in the bottom-right corner and navigates toward the red sphere (goal). (b) A policy trained via BC successfully executes a U-turn to reach the target. (c) However, when the box constraint is applied by projection, the sphere struggles to make the sharp U-turn and ends up colliding with the wall.

the picture is extracted from Action constraint imitation learning (https://openreview.net/pdf?id=UIAkM88Vum)

In real-world reinforcement learning (RL) applications, agents often operate under action constraints arising from physical, safety, or resource limitations. This becomes particularly important in imitation learning (IL), where agents aim to mimic expert behavior but may lack the ability to reproduce certain actions due to mismatched capabilities.

In online, model-based IL, these constraints must be respected during both learning and deployment. Unlike offline settings, the agent must learn safely while interacting with the environment in real time, requiring action feasibility to be enforced dynamically. If constraints are ignored, the agent may execute infeasible actions, leading to unsafe behaviors or task failure.

Diccusion : why do we seperate test-time constraint and test-time constraint

We separate no constraint, test-time constraint, and train-test-time constraint to isolate the effects of action feasibility at different stages. This helps us understand whether constraints should influence learning, execution, or both. Applying constraints only at test time reveals how well an unconstrained policy adapts post hoc, while train-time constraints test whether learning under limited actions hinders policy expressiveness. Combining both shows the cumulative effect. This separation is crucial for designing agents that remain performant without over-restricting their learning dynamics.

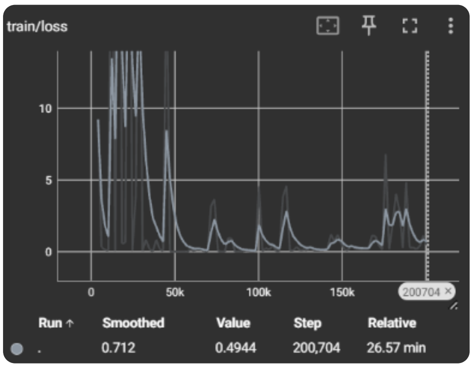
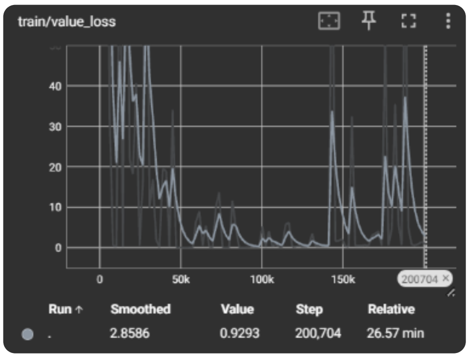
Task2 : Action constraint in model-free

Youtube : link1 (https://youtu.be/WvdVmw0-MBw), link2 (https://youtu.be/v1vu3WKLKCc), link3 (https://youtu.be/uC1gjh0sIbQ)

PPO

Training

no constraint

loss	value loss
 Run ↑ Smoothed Value Step Relative ● . 0.712 0.4944 200,704 26.57 min	 Run ↑ Smoothed Value Step Relative ● . 2.8586 0.9293 200,704 26.57 min

constraint (ignore)

evaluation result

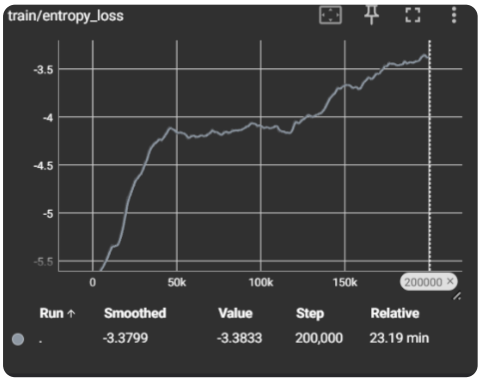
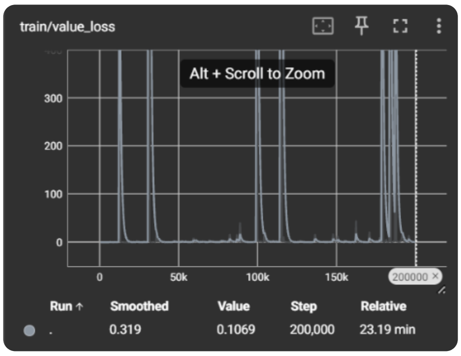
constraint	no constraint	test time constraint	train-test time constraint
ep_rew_mean	191.62	147.92	100.58

A2C

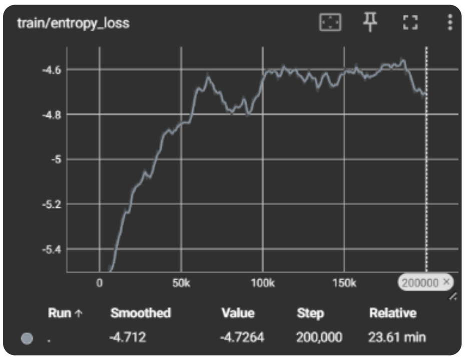
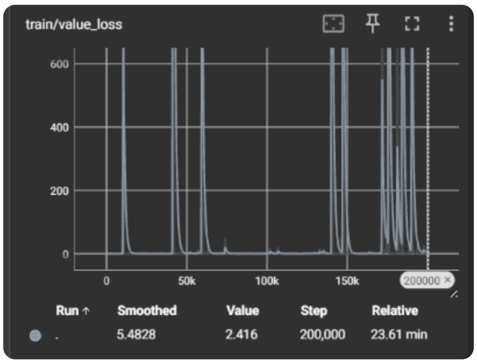
Youtube : link1 (https://youtu.be/I6WiWrqCPtc), link2 (https://youtu.be/7oam9MmmfYU), lik3 (https://youtu.be/Gj6CWdoxB5o)

Training : Bipedal Walker

no constraint

entropy loss	value loss
 Run ↑ Smoothed Value Step Relative . -3.3799 -3.3833 200,000 23.19 min	 Run ↑ Smoothed Value Step Relative . 0.319 0.1069 200,000 23.19 min

constraint

entropy loss	value loss
 Run ↑ Smoothed Value Step Relative . -4.712 -4.7264 200,000 23.61 min	 Run ↑ Smoothed Value Step Relative . 5.4828 2.416 200,000 23.61 min

evaluation result

constraint	no constraint	test time constraint	train-test time constraint
ep_rew_mean	-102.08	-101.57	-98.51

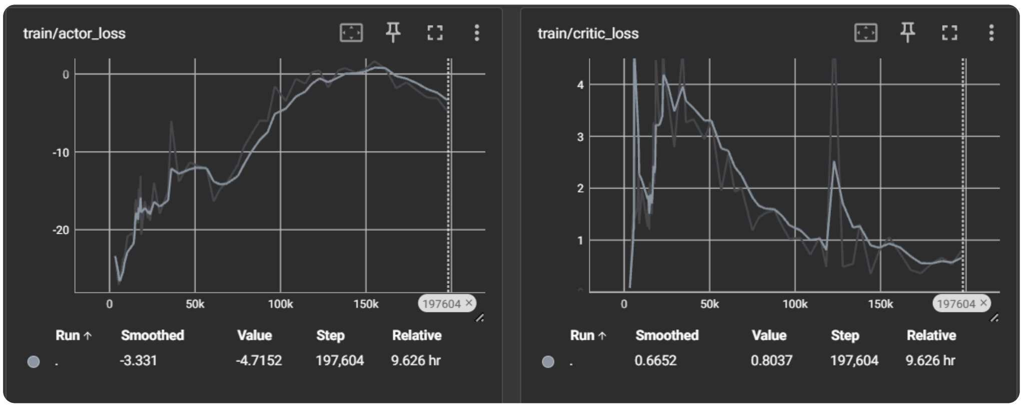
SAC

Training : Bipedal Walker

Youtube : link1 (https://youtu.be/YNicShLri-g), link2 (https://youtu.be/sLK-DjrtSDc), link3 (https://youtu.be/gKhsd-Q70kk)

no constraint (no matter since the training is successful)

constraint



Run ↑	Smoothed	Value	Step	Relative
● .	-3.331	-4.7152	197,604	9.626 hr

Run ↑	Smoothed	Value	Step	Relative
● .	0.6652	0.8037	197,604	9.626 hr

evaluation result

constraint	no constraint	test time constraint	train-test time constraint
ep_rew_mean	287.55	239.95	284.28

Diccusion

1. Why PPO Succeeds While A2C Fails

PPO outperforms A2C largely due to its clipped surrogate objective and minibatch updates, which reduce variance and lead to stable training. In contrast, A2C relies on standard policy gradients, making it more sensitive to noisy value estimates and unstable learning. As a result, PPO consistently achieves positive rewards, while A2C struggles to learn effective policies in long-horizon environments like BipedalWalker.

2. Why PPO Performs Worse Under Constraints and Why Test-Time Constraint > Train-Test Constraint

PPO's performance drops under action constraints, especially when applied during both training and testing. Test-time-only constraints allow the policy to explore freely during learning and adapt later, resulting in better outcomes. However, train-time constraints restrict the agent’s action space early on, reducing policy expressiveness and exploration, which leads to conservative and less effective strategies.

3. why SAC performs the best

SAC achieves the best performance due to its off-policy learning and entropy-regularized objective, which promotes both exploration and stability. Its ability to learn from diverse experiences and maintain stochastic policies helps it adapt well under constraints. Even when the action space is limited, SAC remains robust by balancing exploitation and exploration, making it well-suited for complex control tasks like BipedalWalker.

Task3 : action constraint in model-based

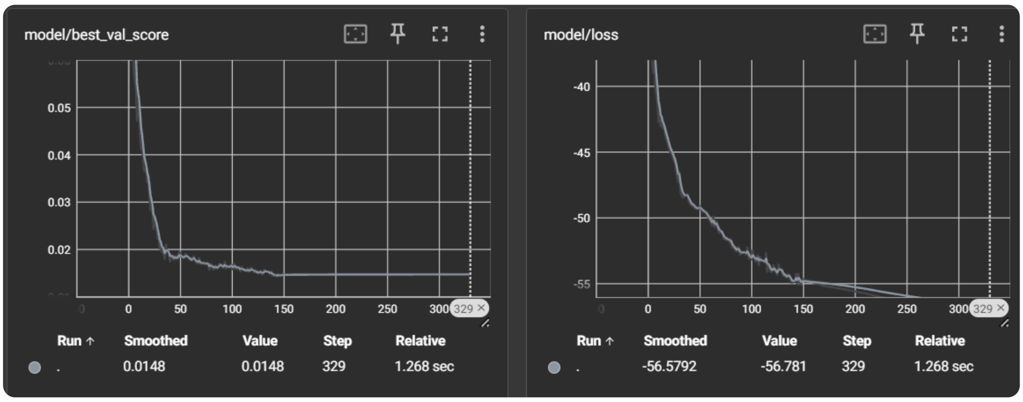
Youtube : link1 (https://youtu.be/CBzVO56gfI4), link2 (https://youtu.be/tORlytB_Nas)

PETS

Training : Bipedal Walker

no constraint (ignore since it is similar to below)

constraint


model/best_val_score

Run ↑	Smoothed	Value	Step	Relative
● .	0.0148	0.0148	329	1.268 sec

model/loss

Run ↑	Smoothed	Value	Step	Relative
● .	-56.5792	-56.781	329	1.268 sec

evaluation result

constraint	no constraint	test time constraint	train-test time constraint
ep_rew_mean	-97.29	-95.97	-98.51

Diccusion : Why PETS fails in Bipedal Walker

In Task 3, we observe that PETS fails to learn a successful policy in BipedalWalker across all constraint settings. A key reason lies in the nature of the environment: although BipedalWalker provides dense reward signals in theory, it behaves like a partially sparse-reward environment in practice. Small or early motor adjustments often yield no immediate feedback, while meaningful rewards are tied to maintaining balance and forward motion over extended time horizons.

This poses a challenge for model-based algorithms like PETS, which rely on accurate short-term dynamics and trajectory sampling to plan optimal actions. In partially sparse settings, the reward signal is delayed or weakly correlated with early state transitions, making it difficult for the learned dynamics model to guide planning effectively. As a result, PETS tends to generate suboptimal plans that fail to stabilize walking, leading to poor and unstable performance.

Task 4:

Youtube : link1 (https://youtu.be/VHm-ZNfOftw), link2 (https://youtu.be/h2xlpDklrUc), link3 (https://youtu.be/h2xlpDklrUc)

GAIL

Training : Bipedal Walker

evaluation result

constraint	no constraint	test time constraint	train-test time constraint
ep_rew_mean	-103.48	-92.12	-94.59

Diccusion : Why GAIL fail in Bipedal Walker

Although GAIL is designed to mimic expert behavior, it fails in BipedalWalker due to a combination of limited expert coverage and sparse learning signals. The expert data, while optimal, only demonstrates a narrow subset of successful trajectories — it does not cover all possible states the agent might visit during exploration. As a result, when the agent deviates from these expert-like paths (which happens frequently early in training), the discriminator can easily distinguish its behavior and assigns near-zero rewards.

This leads to sparse feedback from the discriminator, especially at the start of training. Since the agent's initial behavior is far from expert-like, it receives little to no learning signal and cannot improve its policy. In long-horizon, unstable environments like BipedalWalker, this lack of early reward is particularly harmful. The agent fails to recover, resulting in poor performance despite access to expert demonstrations.

Conclusion

In this study, I find that model-free methods outperform model-based and imitation learning under action constraints. Model-based algorithms struggle with delayed or sparse rewards, while imitation learning like GAIL suffers from sparse feedback early in training. However, algorithms with strong exploration, like SAC, handle constraints better by maintaining robustness and adaptability. This highlights the importance of effective exploration in constrained or sparse-reward environments.

References

- stable-baselines3 documentation (https://stable-baselines3.readthedocs.io/)
- mbrl library (https://github.com/facebookresearch/mbrl-lib)
- OpenReview paper on action-constrained IL (https://openreview.net/pdf?id=UIAkM88Vum)