CS687 Final Project

Anh Pham, Trung Nguyen

1 Introduction

Policy gradient methods directly optimize parameterized policies to maximize expected return, offering natural handling of stochastic policies and continuous action spaces. However, these methods suffer from high gradient variance, which can severely impair learning. Two prominent approaches address this challenge through different mechanisms: REINFORCE with Baseline uses learned state-value functions to reduce variance in Monte Carlo updates, while Actor-Critic methods employ temporal-difference learning for immediate, bootstrapped updates. We conduct a systematic comparison of REINFORCE with Baseline and Actor-Critic on two standard benchmarks: CartPole-v1 and MountainCarContinuous-v0.

2 Environments

2.1 Continuous Mountain Car Environment

The Continuous Mountain Car domain (Moore, 1990) is a standard benchmark for continuous-control reinforcement learning. An underpowered car must drive up a steep right hill, but the engine alone cannot reach the goal; the agent must first drive in the opposite direction to build momentum. This long-horizon credit assignment problem makes the environment a useful testbed for policy-gradient and value-based methods. We use the `MountainCarContinuous-v0` implementation from OpenAI Gym (Towers et al., 2024). We did not choose to go with the same environment in HW2 (further explanation is in Appendix A)

a) State space ($\mathcal{S}$). Each state is a tuple $(x, \dot{x})$, where $x \in [-1.2, 0.6]$ is the car's horizontal position and $\dot{x} \in [-0.07, 0.07]$ is its velocity. Episodes start near $x \approx -0.5$ with zero velocity. The continuous state space requires function approximation rather than tabular methods.

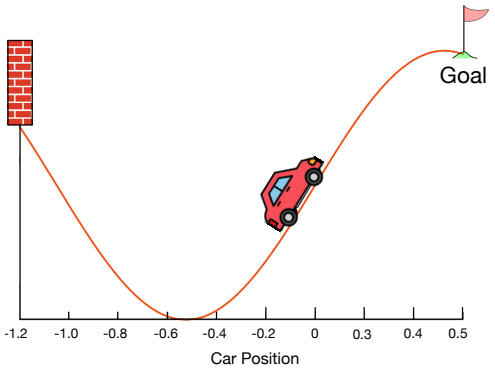


Figure 1: The Mountain Car domain. Image taken from HW2.

b) Action space ($\mathcal{A}$). The agent applies a continuous force $a \in [-1.0, 1.0]$, where negative values accelerate the car to the left and positive values to the right. This contrasts with the discrete 3-action variant used in class.

c) Dynamics. Given state $(x_t, \dot{x}_t)$ and action a_t, the next state is

$$\dot{x}_{t+1} = \dot{x}_t + 0.001\, a_t - 0.0025 \cos(3x_t),$$

$$x_{t+1} = x_t + \dot{x}_{t+1}.$$

Velocity is clipped to $[-0.07, 0.07]$ and position to $[-1.2, 0.6]$. If the car hits the left boundary ($x = -1.2$), its velocity is reset to zero.

d) Reward function ($\mathcal{R}$). At each step the agent receives

$$R_{t+1} = -0.1 a_t^2,$$

a small control penalty that encourages efficient actions. An additional reward of $+100$ is given when the car reaches the goal region $x \geq 0.45$. Most rewards are therefore small and negative, with sparse positive feedback at the goal.

e) Episode termination. Episodes terminate when the goal is reached or when a maximum number of time steps is exceeded. Successful policies

must exploit the nonlinear dynamics to build momentum and reach the goal under sparse reward signals.

2.2 CartPole

The CartPole domain (Barto et al., 1983) is a classic control benchmark in which an agent must balance a pole attached to a cart by applying left or right forces. The task is simple to visualize but captures key reinforcement learning challenges, and we use the standard OpenAI Gym implementation (Towers et al., 2024).

a) State space ($\mathcal{S}$). Each state is a 4-dimensional vector $(x, \dot{x}, \theta, \dot{\theta})$, where x is the cart position, $\dot{x}$ its velocity, θ the pole angle, and $\dot{\theta}$ its angular velocity. Episodes begin near the center with small random perturbations.

b) Action space ($\mathcal{A}$). The agent selects one of two discrete actions: push the cart left or push it right. This contrasts with Mountain Car's continuous action space and requires the agent to learn precise timing rather than graded control.

c) Dynamics. The environment simulates the Newtonian dynamics of an inverted pendulum. State variables are updated deterministically according to these equations, and position and angle are clipped to their allowable ranges. If any limit is exceeded, the episode terminates.

d) Reward function ($\mathcal{R}$). At each timestep the agent receives:

$$R_t = 1,$$

as long as the pole remains within the allowed angle and the cart stays on the track. Thus the return corresponds to the number of steps the pole stays balanced, with a maximum episode length of 500.

e) Episode termination. Episodes end when:

- the pole angle exceeds the allowed threshold,
- the cart position moves beyond the track limits, or
- 500 steps have elapsed.

f) Significance. CartPole is lightweight, deterministic, and highly reproducible, making it one of the most widely used benchmarks for evaluating stability and learning speed of reinforcement learning algorithms.

2.3 REINFORCE with Baseline

REINFORCE with Baseline (Williams, 1992) (Algorithm 1) is a Monte Carlo policy-gradient method that updates the policy using full-episode returns. The policy is parameterized by $\pi_\theta(a \mid s)$, and after generating an episode, the algorithm computes the discounted return

$$G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$$

for every timestep.

Using full returns introduces high variance, so a baseline is used to reduce it. We implement the baseline as a learned value function $\hat{v}(s, w)$ and form an advantage estimate

$$A_t = G_t - \hat{v}(s_t, w).$$

The baseline is trained by minimizing the squared error between predicted values and observed returns. The policy parameters are updated in the direction suggested by the advantage-weighted log-probability gradient.

Algorithm 1 REINFORCE with Baseline (Episodic)

Inputs: Policy $\pi(a \mid s, \theta)$, value function $\hat{v}(s, w)$
Parameters: Learning rates α^θ, α^w
Initialize θ and w
while true **do** ▷ Each iteration is one episode
  Generate episode $S_0, A_0, R_1, \ldots, S_T$
  for each timestep $t = 0, \ldots, T-1$ **do**
    $G_t \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ ▷ Return
    $\delta \leftarrow G_t - \hat{v}(S_t, w)$ ▷ Advantage
    $w \leftarrow w + \alpha^w \, \delta \, \nabla_w \hat{v}(S_t, w)$
    $\theta \leftarrow \theta + \alpha^\theta \, \gamma^t \delta \, \nabla_\theta \log \pi(A_t \mid S_t, \theta)$
  end for
end while

2.4 Actor–Critic

Actor–Critic methods (Algorithm 2) combine policy-gradient updates with value estimation. The critic learns a value function $\hat{v}(s, w)$, and the actor updates the policy parameters θ using feedback from the critic. After each transition, the critic computes a one-step temporal-difference (TD) error:

$$\delta = R + \gamma \hat{v}(S', w) - \hat{v}(S, w),$$

which serves both as the value-function update signal and as an estimate of the advantage for the

policy update. Because updates occur at every time step, Actor–Critic has lower variance than Monte Carlo methods like REINFORCE and typically learns more efficiently in long episodes.

Algorithm 2 One-step Actor–Critic (Episodic)

Inputs: Policy $\pi(a \mid s, \theta)$, value function $\hat{v}(s, w)$
Parameters: Learning rates α^θ, α^w
Initialize θ and w
while true **do** ▷ Each iteration is one episode
  Initialize starting state S
  $I \leftarrow 1$ ▷ Discount factor accumulator
  while S is not terminal **do**
    Sample action $A \sim \pi(\cdot \mid S, \theta)$
    Take action A, observe reward R and next state S'
    if S' is terminal **then**
      $\hat{v}(S', w) \leftarrow 0$
    end if
    $\delta \leftarrow R + \gamma \hat{v}(S', w) - \hat{v}(S, w)$ ▷ TD error
    $w \leftarrow w + \alpha^w \, \delta \, \nabla_w \hat{v}(S, w)$
    $\theta \leftarrow \theta + \alpha^\theta \, I \delta \, \nabla_\theta \log \pi(A \mid S, \theta)$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$
  end while
end while

3 Experiment

3.1 Shared Experimental Configuration

Both REINFORCE and Actor–Critic were evaluated under matched settings to allow a fair comparison. Table 1 lists the shared experimental parameters.

Table 1: Shared Experimental Configuration

Parameter	CartPole	MountainCarContinuous
Episodes	1000	1000
Seeds	5	5
Hidden Dimension	64	64
Discount Factor (γ)	0.99	0.99

All models use fully connected neural networks for both the policy and value functions. For CartPole, both algorithms use two hidden layers with 64 units and ReLU activations. For MountainCarContinuous, REINFORCE uses the same architecture, while Actor–Critic uses ELU activations. We use ELU because MountainCar's state distribution contains sharp nonlinearities, and ELU reduces saturation effects that often cause unstable gradients during training.

The policy head depends on the action space: (1) a softmax distribution for CartPole's two actions, and (2) a Gaussian distribution (mean and log-standard deviation) for Continuous MountainCar. The critic mirrors the policy network and outputs a scalar value estimate. Adam is used for optimization with independent learning rates α^θ (actor) and α^w (critic).

Mountain Car states are normalized to $[-1, 1]$ before being passed into the networks to improve numerical stability.

3.2 Hyperparameter Selection Strategy

Because this is a course project, compute time and available hardware influenced our hyperparameter evaluation strategy. The REINFORCE and Actor–Critic experiments were run by different team members, resulting in slightly different approaches:

- **REINFORCE:** Due to limited compute, CartPole hyperparameters were first screened for 500 episodes across 5 seeds, and performance was measured using the average return of the final 100 episodes. MountainCarContinuous required longer training for meaningful evaluation, so each learning-rate configuration was trained for the full 1000 episodes, but using only 3 seeds.
- **Actor–Critic:** With more compute available (Unity Cluster), we ran all hyperparameter combinations directly for the full 1000 episodes across 5 seeds. No reduced sweep phase was used.

This setup reflects practical constraints: CartPole learns quickly and can be screened with shorter runs, while MountainCar requires long training even for preliminary evaluation. The final models for both algorithms were trained for 1000 episodes with 5 seeds.

3.3 REINFORCE Experiments

We evaluated REINFORCE with a learned baseline on CartPole-v1 and MountainCarContinuous-v0.

3.3.1 CartPole

Hyperparameter Sweep. We tested policy learning rates $\alpha^\theta \in \{10^{-4}, 3 \times 10^{-4}, 10^{-3}, 3 \times 10^{-3}\}$ and value-function learning rates $\alpha^w \in \{10^{-3}, 3 \times$

$10^{-3}, 10^{-2}\}$, trained for 500 episodes across 5 seeds. The best configuration was then retrained for 1000 episodes using 5 seeds.

Final Setting. $\alpha_\theta = 3 \times 10^{-3}$, $\alpha_w = 3 \times 10^{-3}$, hidden dimension = 64.

3.3.2 MountainCar

Hyperparameter Sweep. Due to the task's difficulty, each configuration was trained for the full 1000 episodes across 3 seeds. We tested $\alpha_\theta \in \{3\times10^{-4}, 5\times10^{-4}, 10^{-3}\}$ and $\alpha_w \in \{10^{-3}, 3\times10^{-3}, 5\times10^{-3}\}$.

Final Setting. $\alpha_\theta = 5 \times 10^{-4}$, $\alpha_w = 10^{-3}$.

3.3.3 Training Procedure

At the end of each episode, the policy was updated using the REINFORCE gradient

$$\nabla_\theta J(\theta) = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\,(G_t - V_w(s_t)),$$

with the value function optimized via mean squared error.

3.4 Actor–Critic Experiments

Actor–Critic updates the policy and value function at every timestep using the TD error $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$. We evaluated a range of learning-rate pairs under the full 1000-episode, 5-seed setting.

Table 2: Actor–Critic Learning Rates Tested

Environment	α_θ	α_w
CartPole-v1	1×10^{-3}	1×10^{-3}
	1×10^{-3}	1×10^{-4}
	1×10^{-4}	1×10^{-3}
	5×10^{-4}	1×10^{-3}
	5×10^{-5}	1×10^{-4}
MountainCarContinuous	1×10^{-2}	1×10^{-2}
	1×10^{-3}	1×10^{-3}
	1×10^{-4}	1×10^{-4}
	1×10^{-5}	1×10^{-5}
	1×10^{-4}	5×10^{-4}

All runs used the architectures described earlier, with ELU activations for MountainCar as noted.

4 Results and Comparison

4.1 Learning Curves

Figure 2 shows the learning curves for the two best REINFORCE configurations on both CartPole and MountainCar. For Actor–Critic, Figures 3 and 4 display the learning curves for all tested hyperparameter settings on the two environments. These plots summarize how each method performs and how sensitive they are to different learning-rate combinations.

4.2 Final Performance

Table 3 summarizes the best possible reward for both algorithms across both environments.

5 Discussion

5.1 Performance of REINFORCE

5.1.1 Learning rate sweep

The results for lr sweep can be seen in Figure 6. For MountainCar, none of the LR settings show meaningful improvement: all configurations remain between -35 and -50 with large fluctuations, and higher rates only increase variance. Smaller learning rates stabilize updates but still do not lead to learning. CartPole behaves differently: several LR choices allow steady improvement during the first 300–400 episodes.

5.1.2 CartPole

With the selected hyperparameters, REINFORCE does make progress on CartPole. Rewards rise steadily until around 400–450 episodes and reach near-optimal levels (500). However, performance drops around 750–800 episodes before partially recovering. The standard deviation is a bit large throughout training, showing high sensitivity to random seeds and unstable gradients.

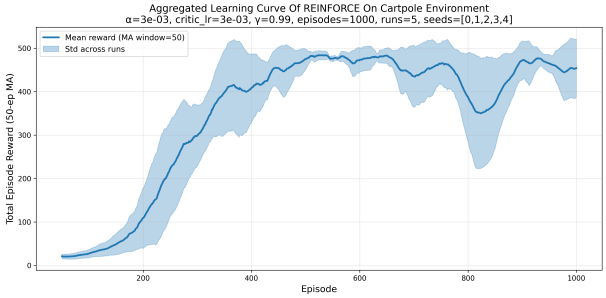
5.1.3 MountainCar

MountainCar remains difficult for REINFORCE. Mean returns remain between -35 and -45, with no steady upward trend across 1000 episodes. Variance across runs is very large and increases over time. The algorithm does not converge reliably under any of the tested settings.

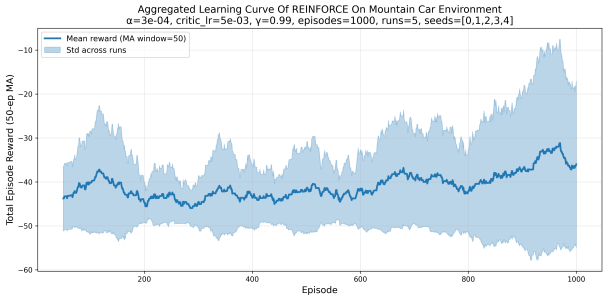
5.2 Performance of Actor–Critic

5.2.1 Cartpole

Across all CartPole experiments, the learning rate had a strong effect on stability. Large actor–critic step sizes ($\alpha^\theta = \alpha^w = 10^{-3}$) learned quickly and peaked around 300-400 episodes, but consistently crashed afterward, with wide standard-deviation bands indicating high variability across seeds. Unbalanced settings with a fast actor and slow critic ($10^{-3}, 10^{-4}$) failed to improve and remained near



((a)) $0.003, 0.003$



((b)) $0.0001, 0.0005$

Figure 2: REINFORCE aggregated learning curves for CartPole and Continuous Mountain Car

Table 3: Final Performance Comparison for REINFORCE and Actor–Critic

Environment	Algorithm	Hyperparameters	Best Reward
CartPole-v1	REINFORCE	$\alpha^\theta = 3 \times 10^{-3},\ \alpha^w = 3 \times 10^{-3}$	494
	Actor–Critic	$\alpha^\theta = 1 \times 10^{-4},\ \alpha^w = 1 \times 10^{-3}$	500
MountainCar-v0	REINFORCE	$\alpha^\theta = 5 \times 10^{-4},\ \alpha^w = 10^{-3}$	-0.09
	Actor–Critic	$\alpha^\theta = 1 \times 10^{-4},\ \alpha^w = 5 \times 10^{-4}$	95.32

10-20 reward throughout. Conversely, smaller actor rates with a larger critic $(10^{-4}, 10^{-3})$ produced the highest and most sustained returns (often 350-450), though still with broad variance. Very small learning rates $(5\times10^{-5}, 10^{-4})$ were the most stable but learned slowly, only reaching 60-85 reward near the end. Overall, higher learning rates led to fast but unstable peaks, while smaller rates resulted in slower yet more reliable improvement.

5.2.2 MountainCar

Across the different learning rates we tested, Actor–Critic showed that Continuous MountainCar is very sensitive to step size. With large learning rates at 0.01, the agent improved at first but quickly collapsed, meaning the updates were too big and caused the policy and value function to destabilize. Medium learning rates like 0.001 were slightly better but still showed a lot of ups and downs.

Smaller learning rates (such as 0.0001) and the mixed setting ($\alpha^\theta = 10^{-4}, \alpha^w = 5 \times 10^{-4}$) produced the highest rewards overall. However, we also noticed that some settings dropped in performance near the end of training. For example, the 0.00001 run improved for several hundred episodes but then declined again near episode 1000, and the mixed setting also dipped late in training. This suggests that even small learning rates can lead to drifting or castastrophic forgetting over long training horizons.

We also observed that the standard deviation across the 5 seeds was very large for almost every learning rate. This means the algorithm's behavior is not very consistent: some runs improve a lot while others get stuck or collapse. Overall, while smaller learning rates give more stable learning curves, all settings show high variability, making MountainCarContinuous a challenging environment for Actor–Critic.

5.3 Comparison between the two methods

We compare the two methods using the best runs reported in Table 3 (See Figure 2 for REINFORCE, and Figure 3c/Figure 4e for Actor–Critic).

Overall, both methods are able to improve performance on CartPole, but REINFORCE tends to learn more smoothly and reaches higher rewards with less variance across runs. In CartPole, Actor-Critic hits near-optimal returns at some episode range, but it shows noticeable instability, especially around 500-700 episodes.

On MountainCar, both algorithms struggle, but Actor–Critic performs slightly better. REINFORCE stays in the range of about -40 to -50 for most of training and shows large variance. Actor–Critic also does not reach the goal reliably, but it achieves higher peaks and shows clearer upward trends, especially in the middle of training. While neither method fully solves the environment, Actor–Critic provides more updates and generally produces higher average returns.

These differences are consistent with the algorithms' update structures. REINFORCE relies on full returns and therefore produces smoother but slower feedback, while Actor–Critic uses bootstrapped TD errors, which can accelerate learning but also introduce instability if the value estimates drift. MountainCar's sparse and delayed rewards favor Actor–Critic's more frequent updates, whereas REINFORCE receives very little learning signal until the agent reaches high position values, which rarely occurs early in training.

In summary, CartPole is a simpler environment where both methods achieve comparably strong performance, with REINFORCE appearing slightly more stable in our runs. In contrast, MountainCar is much harder to learn, leading to high variance and generally low returns for both methods. The high variance indicates that learning is strongly affected by the random seed, and neither method is able to produce consistently good results even under the same hyperparameters.

6 Conclusion

We compared REINFORCE and Actor–Critic on CartPole and MountainCar using matched architectures and controlled experimental settings. Both methods learned effective policies on CartPole, with REINFORCE showing smoother learning and Actor–Critic occasionally achieving optimal returns but with higher instability. On MountainCar, neither method fully solved the task, but Actor–Critic achieved higher peak performance due to its more frequent TD updates. Overall, the results highlight the tradeoff between variance reduction and stability in policy-gradient methods and illustrate how environment difficulty and reward sparsity affect algorithm performance.

References

Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. 1983. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846.

Andrew William Moore. 1990. Efficient memory-based learning for robot control. Technical report, University of Cambridge.

Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, and 1 others. 2024. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*.

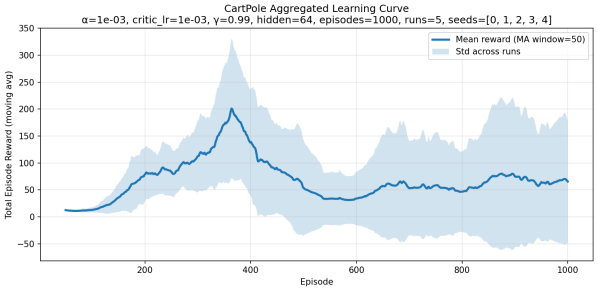
Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256.

A Mountain Car Env

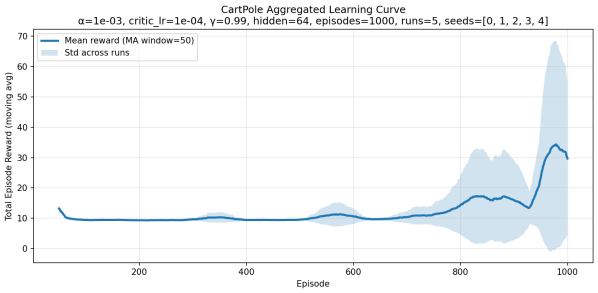
Our earlier homework 2 implemented a fully deterministic version of MountainCar in which the dynamics contain no stochasticity and the reward is a constant -1 until the goal is reached. While this environment is useful for understanding value iteration and planning, we found that it was surprisingly difficult to train both REINFORCE and Actor–Critic on this version of the task. Figure 5 shows one of our runs with -1000 reward for more than 2000 episode using Actor–Critic.

One likely reason is that the deterministic environment provides essentially no informative gradient signal until the agent reaches the exact goal position, making exploration extremely hard for stochastic policy-gradient methods. For this reason, we conduct our experiments on the standard `MountainCarContinuous-v0` environment, which includes smoother dynamics and shaped rewards that provide a more learnable signal for policy-gradient methods.

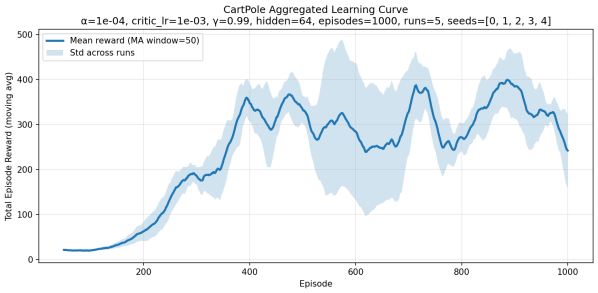
B Learning Rate Sweep



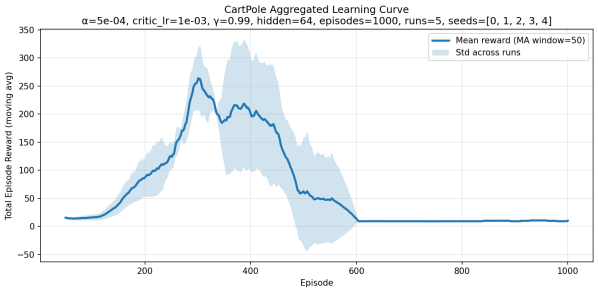
((a)) $\alpha_\theta = 0.001, \alpha_w = 0.001$



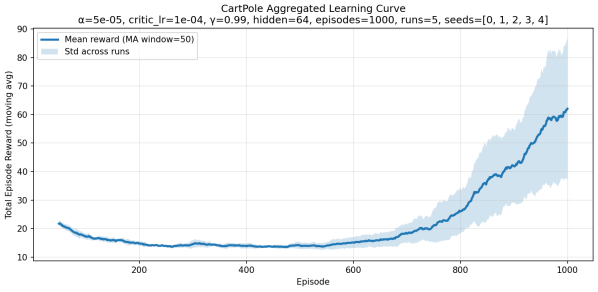
((b)) $0.001, 0.0001$



((c)) $0.0001, 0.001$

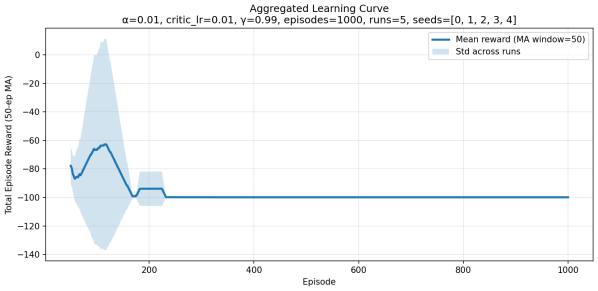


((d)) $0.0005, 0.001$

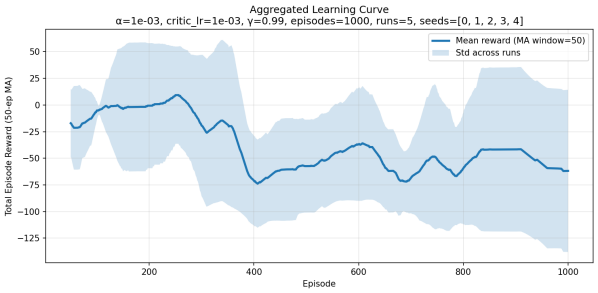


((e)) $0.00005, 0.0001$

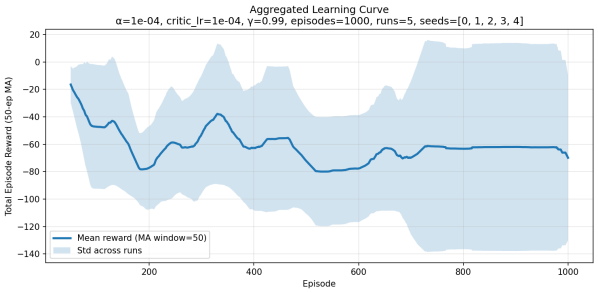
Figure 3: Actor–Critic aggregated learning curves for CartPole.



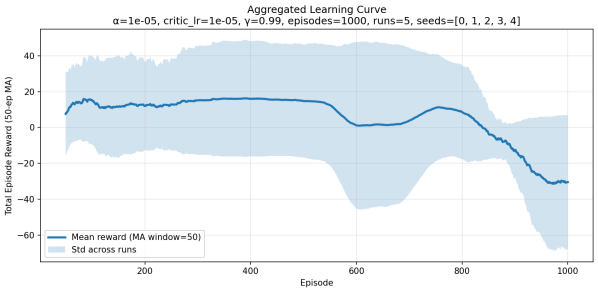
((a)) $\alpha_\theta = 0.01, \alpha_w = 0.01$



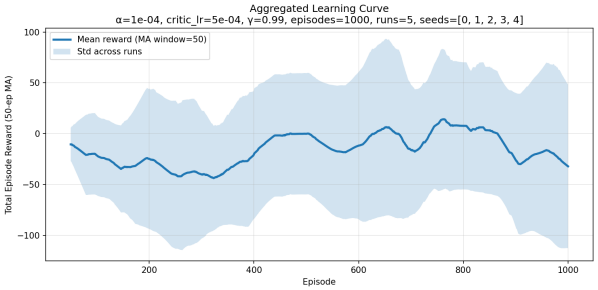
((b)) $0.001, 0.001$



((c)) $0.0001, 0.0001$



((d)) $0.00001, 0.00001$



((e)) $0.0001, 0.0005$

Figure 4: Actor–Critic aggregated learning curves for Continuous MountainCar.

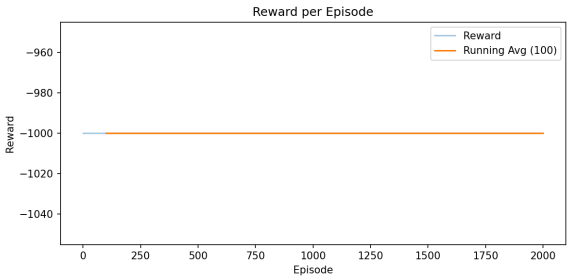
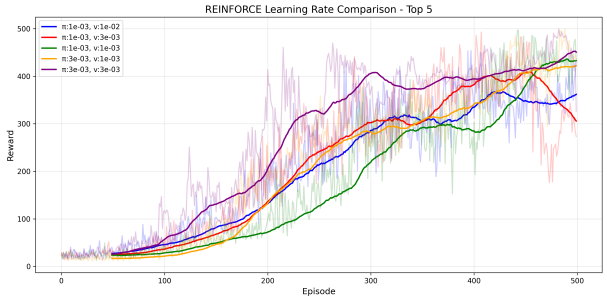
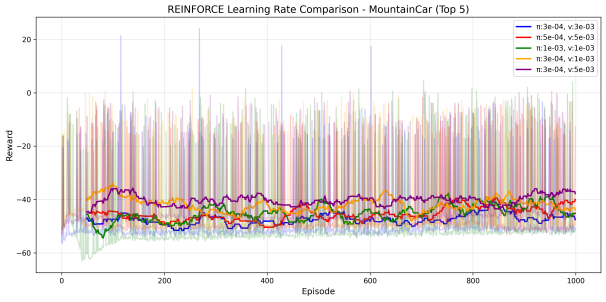


Figure 5: Training curves for Actor–Critic on the deterministic MountainCar environment with $\alpha^{\theta} = 0.0001, \alpha^{w} = 0.0005$ in 2000 episode



(a) CartPole-v1 with REINFORCE



(b) MountainCarContinuous-v0 with REINFORCE

Figure 6: Learning Rate Sweep Comparison Among Set Of Hyperparameters.