CS 285 Assignment 2: Policy Gradients Report

Xiaolei Chu (Student ID: 3038525739)

February 25, 2026

1 Experiment 1: CartPole (Section 3.2)

1.1 Learning Curves

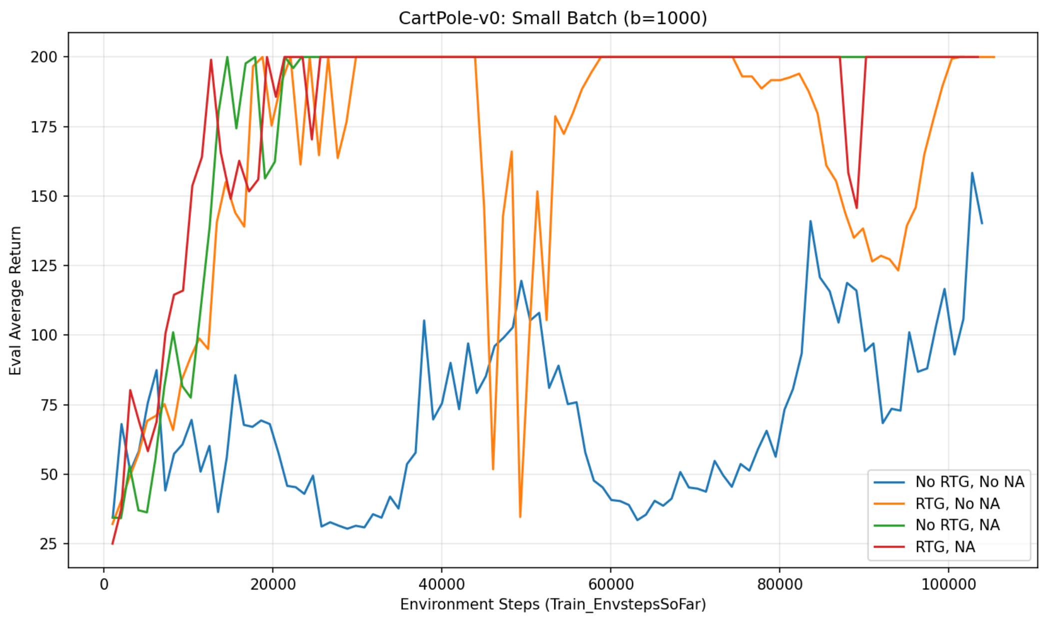

Figure 1: CartPole-v0 learning curves with small batch size ($b = 1000$). The x-axis is the number of environment steps (`Train_EnvstepsSoFar`).

All configurations in both large and small batch settings converge to the maximum return of 200, except for the vanilla configurations without reward-to-go and without advantage normalization, which show the worst performance: in the small batch setting, vanilla (No RTG, No NA) only reaches ~140 at the end, and in the large batch setting, vanilla (No RTG, No NA) only reaches ~102.

1.2 Analysis Questions

Which value estimator has better performance without advantage normalization: the trajectory-centric one, or the one using reward-to-go?

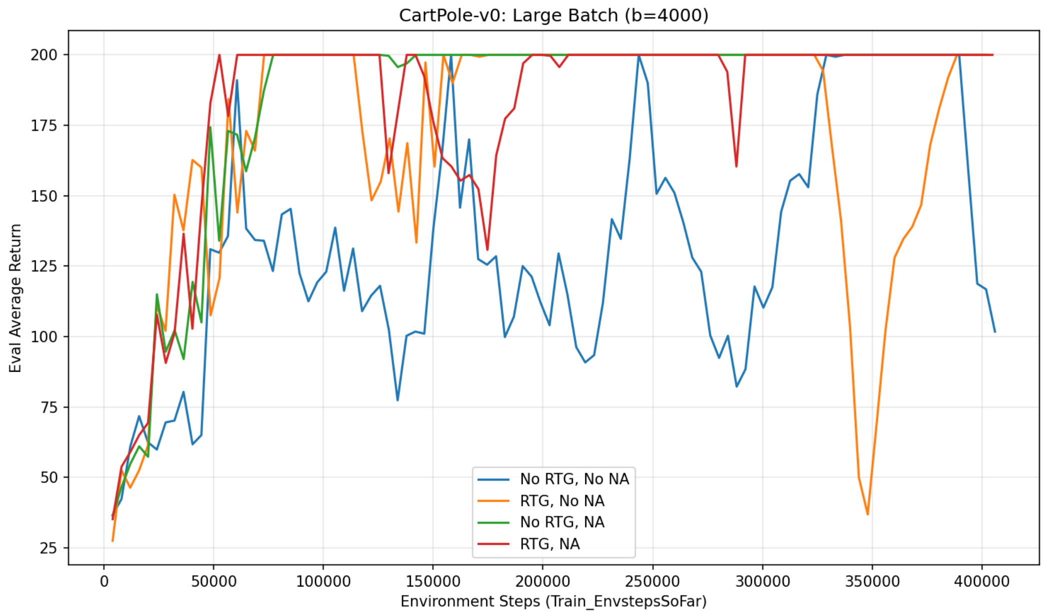

Figure 2: CartPole-v0 learning curves with large batch size ($b = 4000$). The x-axis is the number of environment steps (`Train_EnvstepsSoFar`).

The reward-to-go (RTG) estimator has better performance without advantage normalization. In the small batch setting, RTG reaches 200 while the trajectory-centric estimator only reaches ∼140. The same pattern holds in the large batch setting. This is expected because RTG reduces variance by removing rewards from past timesteps that cannot be affected by the current action.

Between the two value estimators, why do you think one is generally preferred over the other?

Reward-to-go is generally preferred because it exploits the causality structure of the MDP: actions at time t can only affect rewards at times $t' \geq t$. By excluding past rewards from the return estimate, RTG reduces variance without introducing any bias. The trajectory-centric estimator includes rewards from the past which act purely as noise in the gradient estimate.

Did advantage normalization help?

Yes. Advantage normalization consistently improves or matches performance across all settings. In particular, it helps the trajectory-centric estimator (No RTG) converge much more reliably to the maximum return of 200. By normalizing to zero mean and unit variance, the gradient updates become more stable regardless of the scale of the returns.

Did the batch size make an impact?

Yes, increasing the batch size from 1000 to 4000 generally improves learning stability and convergence speed in terms of number of iterations. With a larger batch, the gradient estimate has lower variance. However, each iteration also collects more data, so the total number of environment steps is higher. The large batch is most helpful for the less effective configurations (e.g., No RTG, No NA), making them more stable.

1.3 Command Line Configurations

```
# Small batch experiments (b=1000)
```

```
uv run src/scripts/run.py --env_name CartPole-v0 -n 100 -b 1000 --exp_name
    cartpole
uv run src/scripts/run.py --env_name CartPole-v0 -n 100 -b 1000 -rtg --
   exp_name cartpole_rtg
uv run src/scripts/run.py --env_name CartPole-v0 -n 100 -b 1000 -na --
   exp_name cartpole_na
uv run src/scripts/run.py --env_name CartPole-v0 -n 100 -b 1000 -rtg -na
   --exp_name cartpole_rtg_na

# Large batch experiments (b=4000)
uv run src/scripts/run.py --env_name CartPole-v0 -n 100 -b 4000 --exp_name
    cartpole_lb
uv run src/scripts/run.py --env_name CartPole-v0 -n 100 -b 4000 -rtg --
   exp_name cartpole_lb_rtg
uv run src/scripts/run.py --env_name CartPole-v0 -n 100 -b 4000 -na --
   exp_name cartpole_lb_na
uv run src/scripts/run.py --env_name CartPole-v0 -n 100 -b 4000 -rtg -na
   --exp_name cartpole_lb_rtg_na
```

2 Experiment 2: HalfCheetah (Section 4.2)

2.1 Learning Curves

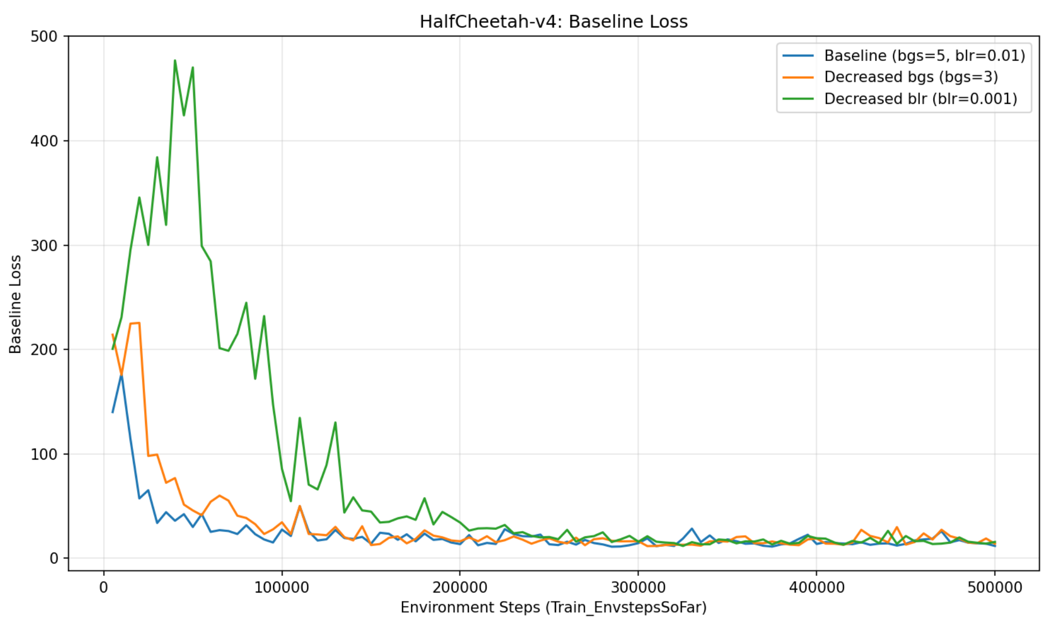

Figure 3: HalfCheetah-v4: Baseline loss curves comparing the default baseline configuration with decreased baseline gradient steps (bgs=3) and decreased baseline learning rate (blr=0.001).

The baselined policy gradient achieves a final eval return of ∼444, well above the threshold of 300. The no-baseline version finishes at ∼-196, demonstrating the significant benefit of using a learned value function baseline for variance reduction.

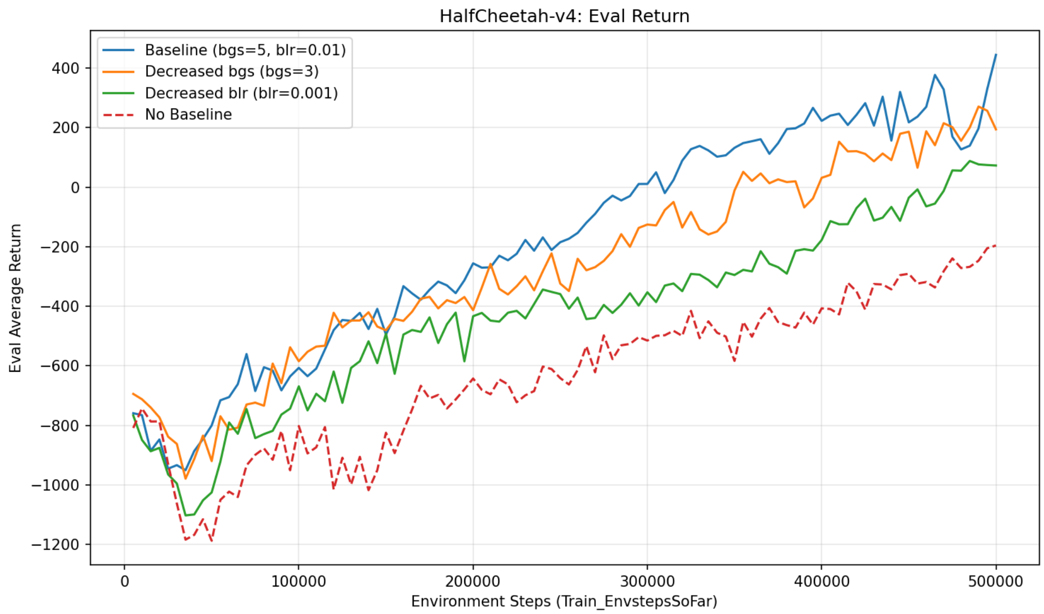

Figure 4: HalfCheetah-v4: Eval return curves. The baselined version (bgs=5, blr=0.01) achieves an average return above 300 at the end of training, while the no-baseline version performs significantly worse.

2.2 Effect of Decreased Baseline Gradient Steps / Learning Rate

We ran two additional experiments varying the baseline hyperparameters:

- **Decreased bgs (bgs=3 vs. default 5):** The baseline loss curve is slightly higher (worse fit), and the final eval return drops to ∼194. With fewer gradient steps, the baseline network cannot fit the value function as accurately, leading to a less effective baseline and noisier policy gradients.

- **Decreased blr (blr=0.001 vs. default 0.01):** The baseline loss remains high throughout training, and the final eval return drops to ∼73. The smaller learning rate severely limits the baseline network's ability to learn, resulting in poor variance reduction and degraded policy performance.

Both modifications hurt performance, confirming that the quality of the baseline fit directly affects policy learning.

2.3 Command Line Configurations

```
# No baseline
uv run src/scripts/run.py --env_name HalfCheetah-v4 -n 100 -b 5000 -eb
    3000 \
  -rtg --discount 0.95 -lr 0.01 --exp_name cheetah

# Baseline (default)
uv run src/scripts/run.py --env_name HalfCheetah-v4 -n 100 -b 5000 -eb
    3000 \
```

```
  -rtg --discount 0.95 -lr 0.01 --use_baseline -blr 0.01 -bgs 5 \
  --exp_name cheetah_baseline

# Decreased baseline gradient steps (bgs=3)
uv run src/scripts/run.py --env_name HalfCheetah-v4 -n 100 -b 5000 -eb
   3000 \
  -rtg --discount 0.95 -lr 0.01 --use_baseline -blr 0.01 -bgs 3 \
  --exp_name cheetah_baseline_decreased_bgs

# Decreased baseline learning rate (blr=0.001)
uv run src/scripts/run.py --env_name HalfCheetah-v4 -n 100 -b 5000 -eb
   3000 \
  -rtg --discount 0.95 -lr 0.01 --use_baseline -blr 0.001 -bgs 5 \
  --exp_name cheetah_baseline_decreased_bgr
```

3 Experiment 3: LunarLander (Section 5)

3.1 Learning Curves

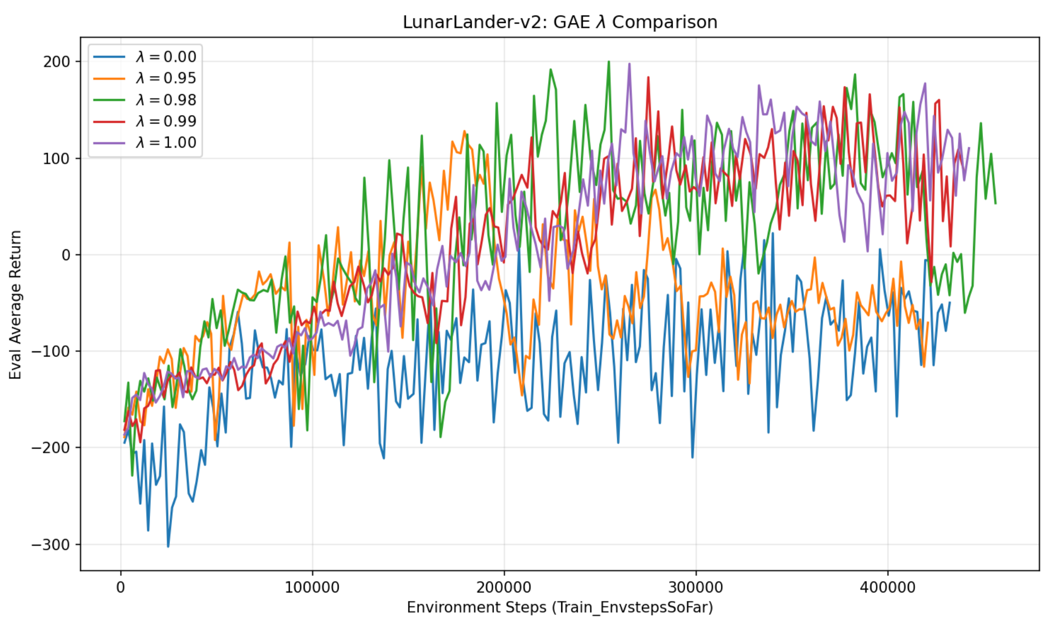

Figure 5: LunarLander-v2: Learning curves for GAE-λ with $\lambda \in \{0, 0.95, 0.98, 0.99, 1\}$.

3.2 Analysis

How did λ affect task performance?

$\lambda = 0$ performs the worst, with max eval return of only ∼22. This is because $\lambda = 0$ corresponds to using only the 1-step TD error δ_t as the advantage estimate, which has high bias. $\lambda = 0.95$ improves significantly (max ∼128) but still does not reach 150. The best performance is achieved by $\lambda = 0.98$ (max ∼200) and $\lambda = 1.0$ (max ∼198), with $\lambda = 0.99$ also performing well (max ∼184).

Overall, higher values of λ lead to better performance on this task, suggesting that variance is less of a concern than bias for LunarLander-v2.

What does $\lambda = 0$ correspond to? What about $\lambda = 1$?

$\lambda = 0$ corresponds to using only the 1-step TD error $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$ as the advantage estimate, which is low variance but high bias. $\lambda = 1$ corresponds to using the full Monte Carlo return minus the baseline (i.e., $Q^\pi(s_t, a_t) - V^\pi(s_t)$), which is unbiased but higher variance. In LunarLander-v2, the high bias of $\lambda = 0$ hurts significantly more than the higher variance of $\lambda = 1$, so larger λ values are preferred.

3.3 Command Line Configurations

```
uv run src/scripts/run.py --env_name LunarLander-v2 --ep_len 1000 \
  --discount 0.99 -n 200 -b 2000 -eb 2000 -l 3 -s 128 -lr 0.001 \
  --use_reward_to_go --use_baseline --gae_lambda <lambda> \
  --exp_name lunar_lander_lambda<lambda>
# where <lambda> in {0, 0.95, 0.98, 0.99, 1}
```

4 Experiment 4: InvertedPendulum (Section 6)

4.1 Learning Curves

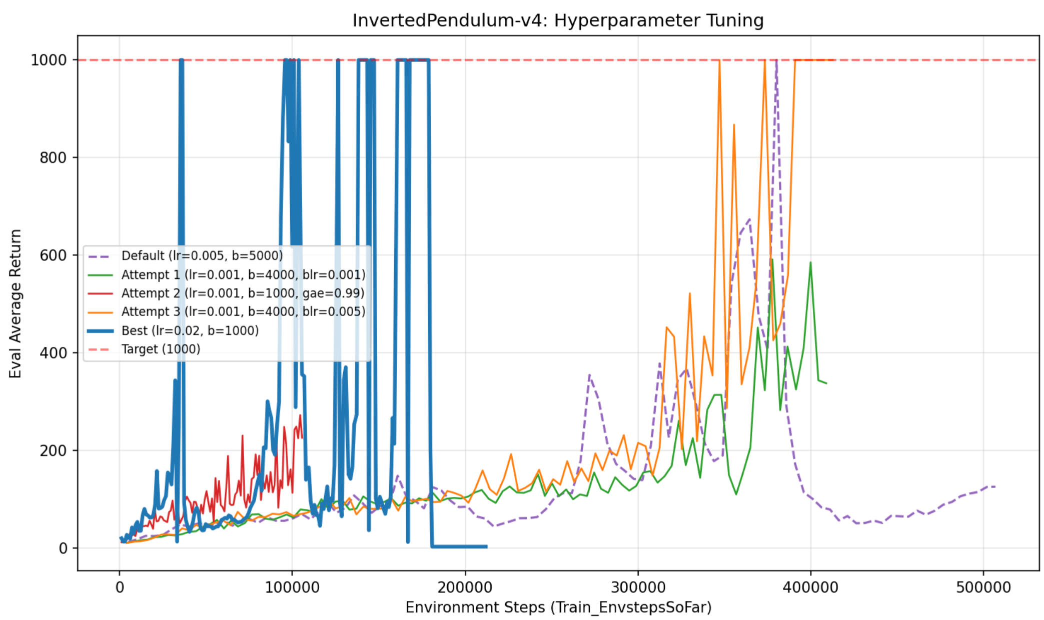

Figure 6: InvertedPendulum-v4: Hyperparameter tuning process. The figure shows the default configuration, three intermediate tuning attempts, and the final best configuration. The best run reaches a return of 1000 within ∼35K environment steps, while the default requires ∼380K steps.

4.2 Tuning Process

We conducted a series of hyperparameter tuning experiments starting from the default configuration. Figure 6 shows the learning curves of all runs. The tuning process proceeded as follows:

1. **Attempt 1** ($lr = 0.001$, $b = 4000$, $blr = 0.001$): Enabled all variance reduction techniques (RTG, baseline, GAE-$\lambda = 0.98$, NA) with a larger network ($s = 128$), but used a conservative policy learning rate and a low baseline learning rate. The poor baseline fit (due to $blr = 0.001$) limited performance to a max return of only $\sim$592.

2. **Attempt 2** ($lr = 0.001$, $b = 1000$, $gae\_\lambda = 0.99$, $bgs = 10$): Reduced batch size for more frequent updates and increased baseline gradient steps to 10. However, the policy learning rate was still too low, resulting in the worst performance (max $\sim$273).

3. **Attempt 3** ($lr = 0.001$, $b = 4000$, $blr = 0.005$): Fixed the baseline learning rate to 0.005, which significantly improved baseline quality. This run reached 1000 but required $\sim$347K steps—still limited by the low policy learning rate.

4. **Best** ($lr = 0.02$, $b = 1000$): The key insight from previous attempts was that the policy learning rate was the bottleneck. Increasing it to 0.02 combined with a small batch size ($b = 1000$) for frequent updates led to reaching 1000 within $\sim$35K steps—a $\sim$10$\times$ improvement over the default.

4.3 Best Hyperparameters

The best configuration differs from the default in the following ways:

- **Learning rate ($lr = 0.02$ vs. default 0.005):** The most impactful change. A higher policy learning rate dramatically sped up convergence.

- **Batch size ($b = 1000$ vs. default 5000):** A smaller batch size enables more frequent policy updates within the same step budget.

- **Reward-to-go, baseline, GAE ($\lambda = 0.98$), and advantage normalization:** All variance reduction techniques were enabled (all disabled by default), providing stable gradient estimates even with the smaller batch and larger learning rate.

- **Discount ($\gamma = 0.99$ vs. default 1.0):** Discounting helps reduce variance in the return estimates.

- **Network size (layer_size= 128 vs. default 64):** A larger network provides more expressive capacity.

4.4 Command Line Configurations

```
# Default configuration (reaches 1000 at ~380K steps)
uv run src/scripts/run.py --env_name InvertedPendulum-v4 -n 100 -b 5000 \
  -eb 1000 --exp_name pendulum

# Attempt 1 (max ~592, lr too low, blr too low)
uv run src/scripts/run.py --env_name InvertedPendulum-v4 -n 100 -b 4000 \
  -eb 1000 -rtg --use_baseline -blr 0.001 -bgs 5 --gae_lambda 0.98 \
```

```
  -na --discount 0.99 -lr 0.001 -s 128 --exp_name pendulum

# Attempt 2 (max ~273, lr too low)
uv run src/scripts/run.py --env_name InvertedPendulum-v4 -n 100 -b 1000 \
  -eb 400 -rtg --use_baseline -blr 0.001 -bgs 10 --gae_lambda 0.99 \
  -na --discount 0.99 -lr 0.001 -s 128 --exp_name pendulum

# Attempt 3 (reaches 1000 at ~347K steps, lr still too low)
uv run src/scripts/run.py --env_name InvertedPendulum-v4 -n 100 -b 4000 \
  -eb 800 -rtg --use_baseline -blr 0.005 -bgs 5 --gae_lambda 0.98 \
  -na --discount 0.99 -lr 0.001 -s 128 --exp_name pendulum

# Best configuration (reaches 1000 at ~35K steps)
uv run src/scripts/run.py --env_name InvertedPendulum-v4 -n 200 -b 1000 \
  -eb 800 -rtg --use_baseline -blr 0.005 -bgs 5 --gae_lambda 0.98 \
  -na --discount 0.99 -lr 0.02 -s 128 --exp_name pendulum
```