CS 285 Homework 2: Policy Gradients

Due September 25, 11:59 pm

3 Policy Gradients

- Create two graphs:
  - In the first graph, compare the learning curves (average return vs. number of environment steps) for the experiments prefixed with `cartpole`. (The small batch experiments.)
  - In the second graph, compare the learning curves for the experiments prefixed with `cartpole_lb`. (The large batch experiments.)

  For all plots in this assignment, the x-axis should be number of environment steps, logged as `Train_EnvstepsSoFar` (*not* number of policy gradient iterations).
- Answer the following questions briefly:
  - Which value estimator has better performance without advantage normalization: the trajectory-centric one, or the one using reward-to-go?
  - Did advantage normalization help?
  - Did the batch size make an impact?
- Provide the exact command line configurations (or `#@params` settings in Colab) you used to run your experiments, including any parameters changed from their defaults.

Solution: I provide the learning curve graphs below:

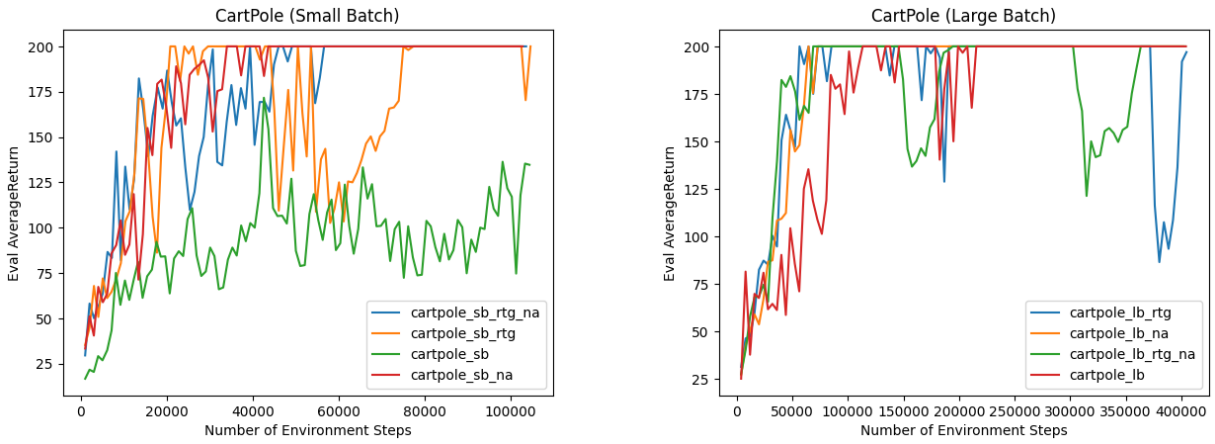


Figure 1: The graph on the left displays the learning curves for the small batch runs, and the right displays those of the large batch runs.

My responses to the questions:

- Without advantage normalization, reward-to-go estimator generally has better performance than the trajectory centric one, as it converges to 200 faster. However, in the large batch experiments, using reward-to-go seems to produce more fluctuations after both reach 200 (e.g. at around environment step 400,000).
- Advantage normalization helps significantly with faster convergence and decreased variance upon convergence.

- The large batch helped the raw (i.e. no RTG nor NA) policy converge to 200 faster.
- The exact command configurations I used are the defaults, provided below:

```
python cs285/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 1000 \
--exp_name cartpole_sb
python cs285/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 1000 \
-rtg --exp_name cartpole_sb_rtg
python cs285/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 1000 \
-na --exp_name cartpole_sb_na
python cs285/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 1000 \
-rtg -na --exp_name cartpole_sb_rtg_na
python cs285/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 4000 \
--exp_name cartpole_lb
python cs285/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 4000 \
-rtg --exp_name cartpole_lb_rtg
python cs285/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 4000 \
-na --exp_name cartpole_lb_na
python cs285/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 4000 \
-rtg -na --exp_name cartpole_lb_rtg_na
```

4 Neural Network Baseline

- Plot a learning curve for the baseline loss.
- Plot a learning curve for the eval return. You should expect to achieve an average return over 300 for the baselined version.
- Run another experiment with a decreased number of baseline gradient steps (`-bgs`) and/or baseline learning rate (`-blr`). How does this affect (a) the baseline learning curve and (b) the performance of the policy?
- **Optional:** Add `-na` back to see how much it improves things. Also, set `video_log_freq 10`, then open TensorBoard and go to the "Images" tab to see some videos of your HalfCheetah walking along!

Solution: I provide the learning curves below:

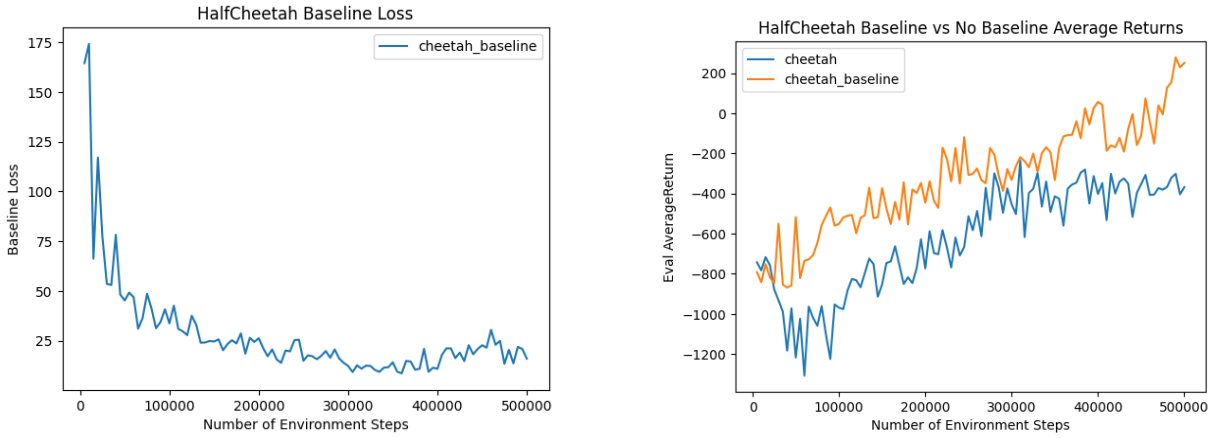


Figure 2: The graph on the left depicts the learning curve for the baseline loss; the one on the right displays learning curve for the average returns (between using a baseline and not using a baseline).

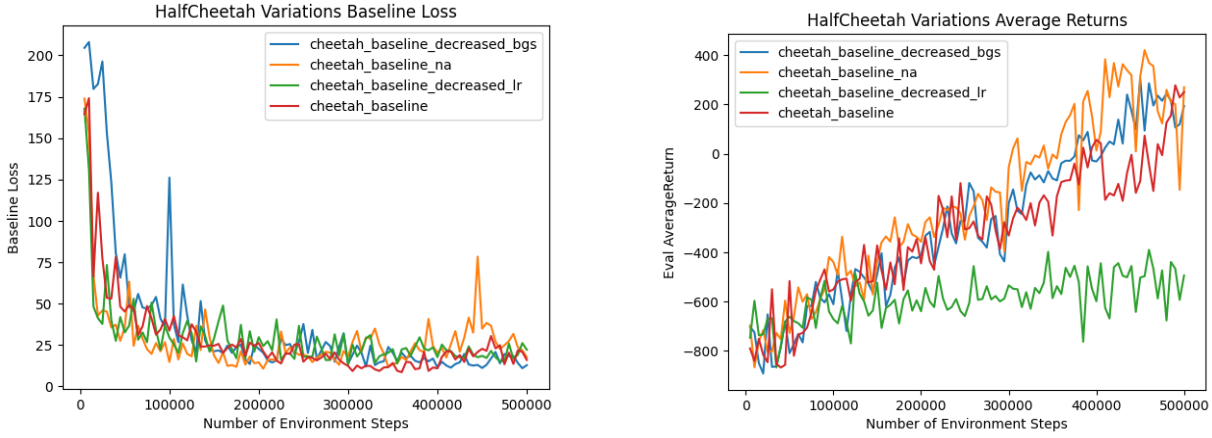


Figure 3: The graph on the left displays the baseline loss curves of the extra experiments, and the right one displays their average return curves.

For the extra experiments, I tried decreasing the number of baseline gradient steps from 5 to 2, and separately the baseline learning rate from 0.01 to 0.001. I also tried adding advantage normalization to see how it improves things.

(a) The decreased learning rate helped the baseline loss converge slightly faster, but the decreased number of gradient steps made the convergence slightly slower.

(b) The decreased learning rate worsened the performance of the policy, while the decreased gradient steps didn't significantly impact the performance convergence nor variance.

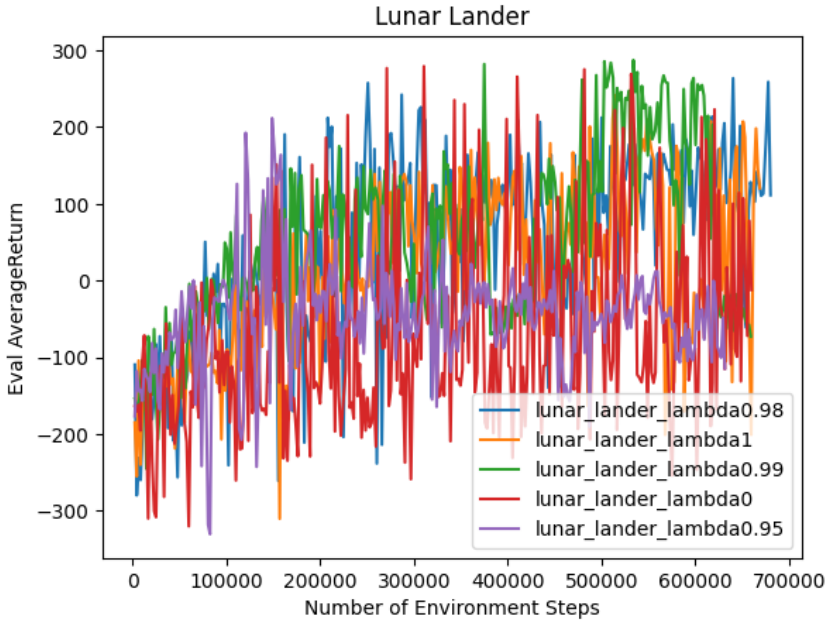
The exact command configurations I used are provided below:

```
# No baseline
python cs285/scripts/run_hw2.py --env_name HalfCheetah-v4 \
-n 100 -b 5000 -rtg --discount 0.95 -lr 0.01 \
--exp_name cheetah

# Baseline
python cs285/scripts/run_hw2.py --env_name HalfCheetah-v4 \
-n 100 -b 5000 -rtg --discount 0.95 -lr 0.01 \
--use_baseline -blr 0.01 -bgs 5 --exp_name cheetah_baseline

# Baseline with advantage normalization
python cs285/scripts/run_hw2.py --env_name HalfCheetah-v4 \
-n 100 -b 5000 -rtg -na --discount 0.95 -lr 0.01 \
--use_baseline -blr 0.01 -bgs 5 --exp_name cheetah_baseline_na

# Baseline with decreased baseline gradient steps
python cs285/scripts/run_hw2.py --env_name HalfCheetah-v4 \
-n 100 -b 5000 -rtg --discount 0.95 -lr 0.01 \
--use_baseline -blr 0.01 -bgs 2 --exp_name cheetah_baseline_decreased_bgs

# Baseline with decreased learning rate
python cs285/scripts/run_hw2.py --env_name HalfCheetah-v4 \
-n 100 -b 5000 -rtg --discount 0.95 -lr 0.001 \
--use_baseline -blr 0.01 -bgs 5 --exp_name cheetah_baseline_decreased_lr
```

5 Generalized Advantage Estimation

- Provide a single plot with the learning curves for the `LunarLander-v2` experiments that you tried. Describe in words how λ affected task performance. The run with the best performance should achieve an average score close to 200 (180+).
- Consider the parameter λ. What does $\lambda = 0$ correspond to? What about $\lambda = 1$? Relate this to the task performance in `LunarLander-v2` in one or two sentences.

Solution: I provide a plot depicting the learning curves for the `LunarLander-v2` experiments below:



The parameter λ represents the balance between prioritizing variance v.s. bias when estimating advantages, in particular through how much we weigh reward accumulation. Thus, $\lambda = 0$ corresponds to estimating advantages solely on the rewards at each step, while $\lambda = 1$ corresponds to estimating advantages based on accumulated rewards.

In regards to the performance in `LunarLander-v2`, we see that the highest λ values (e.g. 0.99 and 1) perform the best, while the lower values of λ perform worse (e.g. 0 and 0.95). In particular, we can see that all the learning curves are relatively similar in the earlier environment steps, but the policies with higher λ values begin to perform better later on since they are able to account for accumulating rewards.

The exact command configurations I used are the defaults, provided below:

```
for lambda in 0 0.95 0.98 0.99 1
do
    python cs285/scripts/run_hw2.py \
        --env_name LunarLander-v2 --ep_len 1000 \
        --discount 0.99 -n 300 -l 3 -s 128 -b 2000 -lr 0.001 \
        --use_reward_to_go --use_baseline --gae_lambda $lambda \
        --exp_name lunar_lander_lambda$lambda
done
```

6 Hyperparameter Tuning

1. Provide a set of hyperparameters that achieve high return on `InvertedPendulum-v4` in as few environment steps as possible.
2. Show learning curves for the average returns with your hyperparameters and with the default settings, with environment steps on the x-axis. Returns should be averaged over 5 seeds.

Solution: The set of hyperparameters that achieve high return on `InvertedPendulum-v4` is: all defaults except return-to-go is disabled and decrease the batch size from 5000 to 3000.

I provide the learning curves below:

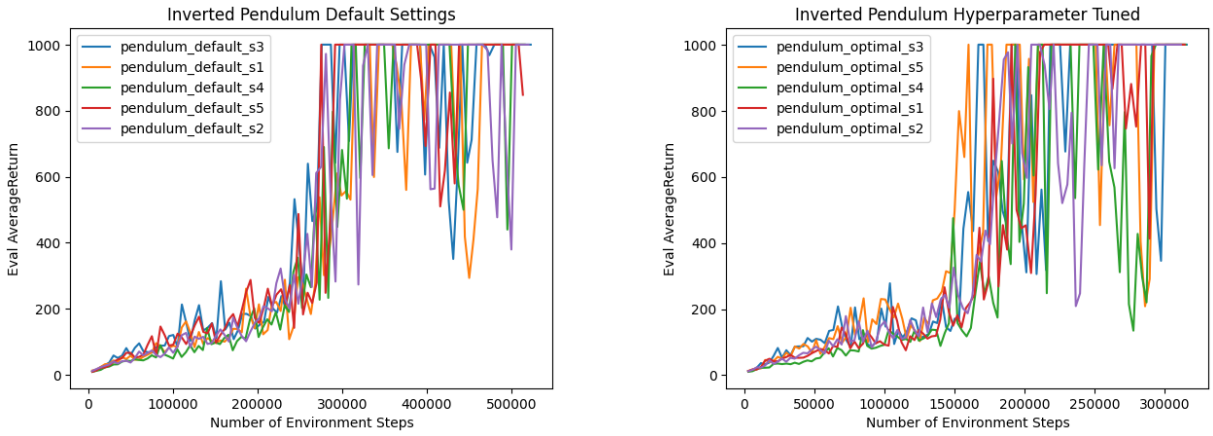


Figure 4: The graph on the left displays the learning curves with the default setting, and the one on the right displays the learning curves with my set of hyperparameters. **We see that my hyperparameter tuning improved the convergence time from around 300,000 environment steps to around around 175,000 environment steps.**

The exact command configurations I used are provided below:

```
# default settings
for seed in $(seq 1 5); do
    python cs285/scripts/run_hw2.py --env_name InvertedPendulum-v4 -n 100 \
        --exp_name pendulum_default_s$seed \
        -rtg --use_baseline -na \
        --batch_size 5000 \
        --seed $seed
done

# my settings
for seed in $(seq 1 5); do
    python cs285/scripts/run_hw2.py --env_name InvertedPendulum-v4 -n 100 \
    --exp_name pendulum_optimal_s$seed \
    --use_baseline -na \
    --batch_size 3000 \
    --seed $seed
done
```

7 (Extra Credit) Humanoid

1. Plot a learning curve for the Humanoid-v4 environment. You should expect to achieve an average return of at least 600 by the end of training. Discuss what changes, if any, you made to complete this problem (for example: optimizations to the original code, hyperparameter changes, algorithmic changes).

Solution: I plot the learning curve for the `Humanoid-v4` environment below:

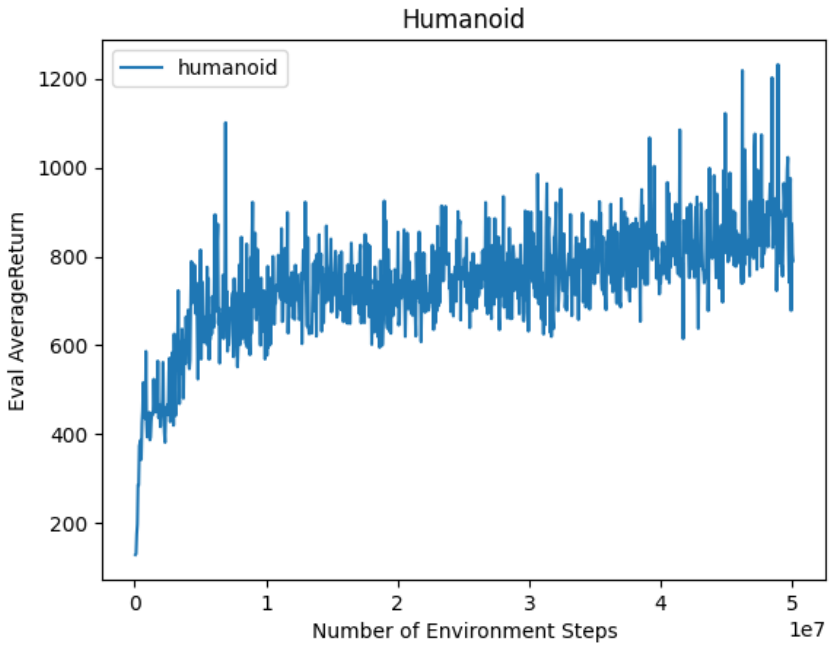


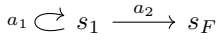
Figure 5: We can see that the policy achieves an average return of far above 600 (i.e. around 900) by the end of training.

The exact command configurations I used are the defaults, provided below:

```
python cs285/scripts/run_hw2.py \
    --env_name Humanoid-v4 --ep_len 1000 \
    --discount 0.99 -n 1000 -l 3 -s 256 -b 50000 -lr 0.001 \
    --baseline_gradient_steps 50 \
    -na --use_reward_to_go --use_baseline --gae_lambda 0.97 \
    --exp_name humanoid --video_log_freq 5
```

8 Analysis

Consider the following infinite-horizon MDP:



At each step, the agent stays in state s_1 and receives reward 1 if it takes action a_1, and receives reward 0 and terminates the episode otherwise. Parametrize the policy as stationary (not dependent on time) with a single parameter:

$$\pi_\theta(a_1|s_1) = \theta, \pi_\theta(a_2|s_1) = 1 - \theta$$

1. Applying policy gradients

   (a) Use policy gradients to compute the gradient of the expected return $R(\tau)$ with respect to the parameter θ. **Do not use discounting.**

   Hint: to compute $\sum_{k=1}^{\infty} k\alpha^{k-1}$, you can write:

$$\sum_{k=1}^{\infty} k\alpha^{k-1} = \sum_{k=1}^{\infty} \frac{d}{d\alpha}\alpha^k = \frac{d}{d\alpha}\sum_{k=1}^{\infty}\alpha^k$$

   Solution: We compute the expected policy gradient as follows:

$$\begin{aligned}
\mathbb{E}_{\pi_\theta}\left[\sum_{t\in\mathcal{T}} \nabla_\theta \log \pi_\theta(a_t \mid s_t) R(\tau)\right] &= \sum_{t=1}^{\infty} \theta^t(1-\theta)\left(\sum_{t'=1}^{t} \nabla_\theta \log \pi_\theta(a_1) + \nabla_\theta \log \pi_\theta(a_2)\right) t \\
&= \sum_{t=1}^{\infty} \theta^t(1-\theta)\left(\frac{t}{\theta} - \frac{1}{1-\theta}\right) t \\
&= \sum_{t=1}^{\infty} \left(\theta^{t-1}(1-\theta)t^2 - \theta^t t\right) \\
&= \sum_{t=1}^{\infty} \left((\theta^{t-1} - \theta^t)\cdot t(t+1) + (-\theta^{t-1} + \theta^t - \theta^t)\cdot t\right) \\
&= (1-\theta)\sum_{t=1}^{\infty} \theta^{t-1} t(t+1) - \sum_{t=1}^{\infty} \theta^{t-1} t \\
&= (1-\theta)\cdot \frac{\partial^2}{\partial\theta^2}\left(\frac{\theta}{1-\theta}\right) - \frac{\partial}{\partial\theta}\left(\frac{1}{1-\theta}\right) \\
&= (1-\theta)\cdot \frac{2}{(1-\theta)^3} - \frac{1}{(1-\theta)^2} \\
&= \frac{1}{(1-\theta)^2}
\end{aligned}$$

(b) Compute the expected return of the policy $\mathbb{E}_{\tau\sim\pi_\theta} R(\tau)$ directly. Compute the gradient of this expression with respect to θ and verify that this matches the policy gradient.

Solution: We compute the expected return directly as follows:

$$\begin{aligned}\mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)] &= \mathbb{E}_{\tau\sim\pi_\theta}[t] \\ &= (1-\theta)\sum_{t=1}^{\infty}\theta^t t \\ &= (1-\theta)\cdot\frac{\theta}{(1-\theta)^2} \\ &= \frac{\theta}{1-\theta}\end{aligned}$$

We then compute the gradient of this expression as follows:

$$\frac{\partial}{\partial\theta}\left(\frac{\theta}{1-\theta}\right) = \frac{1}{(1-\theta)^2},$$

and confirm that it matches the policy gradient computed in part (a).

2. Compute the variance of the policy gradient in closed form and describe the properties of the variance with respect to θ. For what value(s) of θ is variance minimal? Maximal? (Once you have an exact expression for the variance you can eyeball the min/max).

   Hint: Once you have it expressed as a sum of terms $P(\theta)/Q(\theta)$ where P and Q are polynomials, you can use a symbolic computing program (Mathematica, SymPy, etc) to simplify to a single rational expression.

   Solution: We compute the variance of the policy gradient as follows:

$$
\begin{aligned}
&\mathrm{Var}_{\pi_\theta}\left(\sum_{t\in\mathcal{T}} \nabla_\theta \log \pi_\theta(a_t \mid s_t) R(\tau)\right) \\
&= \mathbb{E}_{\pi_\theta}\left[\left(\sum_{t\in\mathcal{T}} \nabla_\theta \log \pi_\theta(a_t \mid s_t) R(\tau)\right)^2\right] - \mathbb{E}_{\pi_\theta}\left[\sum_{t\in\mathcal{T}} \nabla_\theta \log \pi_\theta(a_t \mid s_t) R(\tau)\right]^2 \\
&= \sum_{t=1}^{\infty} \theta^t (1-\theta)\left(\sum_{t'=1}^{t} \nabla_\theta \log \pi_\theta(a_1) + \nabla_\theta \log \pi_\theta(a_2)\right)^2 t^2 - \left(\frac{1}{(1-\theta)^2}\right)^2 \\
&= \sum_{t=1}^{\infty} \theta^t (1-\theta)\left(\frac{t}{\theta} - \frac{1}{1-\theta}\right)^2 t^2 - \frac{1}{(1-\theta)^4} \\
&= \sum_{t=1}^{\infty} \theta^t (1-\theta)\left(\frac{t^4}{\theta^2} - \frac{2t^3}{\theta(1-\theta)} + \frac{t^2}{(1-\theta)^2}\right) - \frac{1}{(1-\theta)^4} \\
&= \frac{4\theta + 9 + \frac{1}{\theta}}{(1-\theta)^4} - \frac{1}{(1-\theta)^4} \\
&= \frac{4\theta + 8 + \frac{1}{\theta}}{(1-\theta)^4}
\end{aligned}
$$

   where between lines 4 and 5, we used a symbolic computing program (Wolfram Alpha) to compute the closed form for the left summation.

3. Apply return-to-go as an advantage estimator.

   (a) Write the modified policy gradient and confirm that it is unbiased.

   Solution: We compute the expected modified policy gradient as follows, where T is the number of steps taken at a_1 before terminating at a_2.

$$\begin{aligned}
\mathbb{E}_{\pi_\theta}\left[\sum_{t\in\mathcal{T}} \nabla_\theta \log \pi_\theta(a_t \mid s_t)(T-t)\right] &= \sum_{T=1}^{\infty} \theta^T(1-\theta) \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t)(T-t) \\
&= \sum_{T=1}^{\infty} \theta^T(1-\theta) \cdot \frac{T(T+1)}{2\theta} \\
&= \frac{1-\theta}{2\theta} \sum_{T=1}^{\infty} \theta^T T(T+1) \\
&= \frac{1}{(1-\theta)^2}
\end{aligned}$$

   where we used a symbolic computing program to simplify the last step. Thus, we confirm that this modified policy gradient is unbiased.

   (b) Compute the variance of the return-to-go policy gradient and plot it on $[0, 1]$ alongside the variance of the original estimator.

   Solution: We compute the variance of the expected return-to-go policy gradient below:

$$\begin{aligned}
&\mathrm{Var}_{\pi_\theta}\left(\sum_{t\in\mathcal{T}} \nabla_\theta \log \pi_\theta(a_t \mid s_t)(T-t)\right) \\
&= \mathbb{E}_{\pi_\theta}\left[\left(\sum_{t\in\mathcal{T}} \nabla_\theta \log \pi_\theta(a_t \mid s_t)(T-t)\right)^2\right] - \mathbb{E}_{\pi_\theta}\left[\sum_{t\in\mathcal{T}} \nabla_\theta \log \pi_\theta(a_t \mid s_t)(T-t)\right]^2 \\
&= \sum_{T=1}^{\infty} \theta^T(1-\theta)\left(\frac{T(T+1)}{2\theta}\right)^2 - \left(\frac{1}{(1-\theta)^2}\right)^2 \\
&= \frac{1-\theta}{4\theta^2} \sum_{T=1}^{\infty} \theta^T T^2(T+1)^2 - \frac{1}{(1-\theta)^4} \\
&= \frac{\theta + 4 + \frac{1}{\theta}}{(1-\theta)^4} - \frac{1}{(1-\theta)^4} \\
&= \frac{\theta + 3 + \frac{1}{\theta}}{(1-\theta)^4}
\end{aligned}$$

   Now, we plot this on [0.05, 0.95] (not [0, 1] so that we can compare better) along with the variance of the original estimator below:

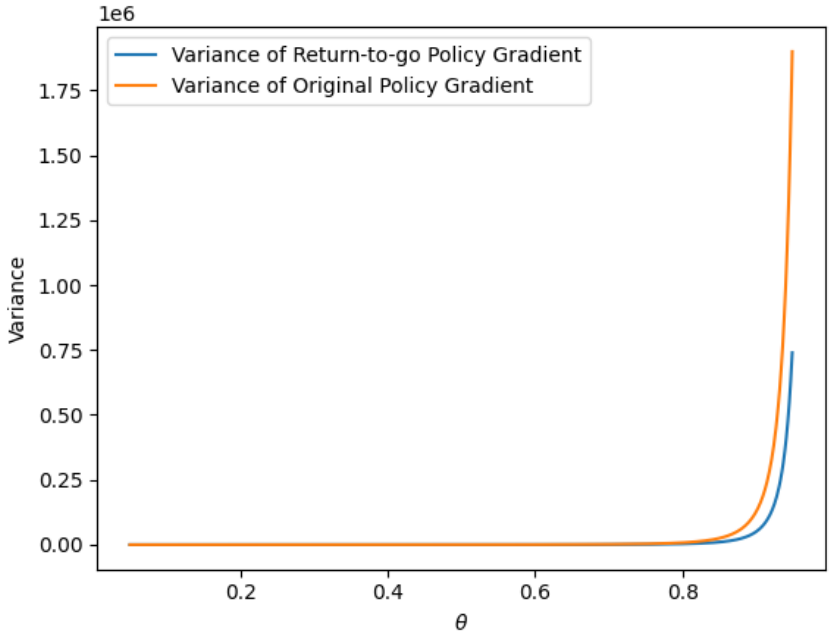
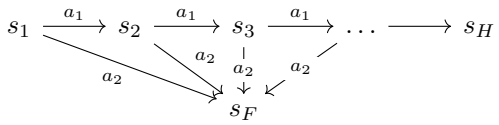


Figure 6: We see that the return-to-go policy controls the variance better.

4. Consider a finite-horizon H-step MDP with sparse reward:



The agent receives reward $R_{\max}$ if it arrives at s_H and reward 0 if it arrives at s_F (a terminal state). In other words, the return for a trajectory τ is given by:

$$R(\tau) = \begin{cases} 1 & \tau \text{ ends at } s_H \\ 0 & \tau \text{ ends at } s_F \end{cases}$$

Using the same policy parametrization as above, consider off-policy policy gradients via importance sampling. Assume we want to compute policy gradients for a policy π_θ with samples drawn from $\pi_{\theta'}$.

(a) Write the policy gradient with importance sampling.

Solution: We compute the expected policy gradient with importance sampling below:

$$\mathbb{E}_{\pi_\theta}\left[\sum_{t\in\mathcal{T}} \nabla_\theta \log \pi_\theta(a_t \mid s_t) R(\tau)\right] = \mathbb{E}_{\pi_{\theta'}}\left[\left(\frac{\theta}{\theta'}\right)^H \frac{H R(\tau)}{\theta}\right] = \theta^{H-1} H$$

(b) Compute its variance.

Solution: We compute the variance of the policy gradient with importance sampling below:

$$
\begin{aligned}
&\mathrm{Var}_{\pi_\theta}\left(\sum_{t\in\mathcal{T}} \nabla_\theta \log \pi_\theta(a_t \mid s_t) R(\tau)\right) \\
&= \mathbb{E}_{\pi_\theta}\left[\left(\sum_{t\in\mathcal{T}} \nabla_\theta \log \pi_\theta(a_t \mid s_t) R(\tau)\right)^2\right] - \mathbb{E}_{\pi_\theta}\left[\sum_{t\in\mathcal{T}} \nabla_\theta \log \pi_\theta(a_t \mid s_t) R(\tau)\right]^2 \\
&= \mathbb{E}_{\pi_{\theta'}}\left[\left(\frac{\theta}{\theta'}\right)^{2H} \frac{H^2 R(\tau)^2}{\theta^2}\right] - \mathbb{E}_{\pi_{\theta'}}\left[\left(\frac{\theta}{\theta'}\right)^{H} \frac{H R(\tau)}{\theta}\right]^2 \\
&= \left(\frac{\theta}{\theta'}\right)^{H} \theta^{H-2} H^2 - \left(\theta^{H-1} H\right)^2 \\
&= \left(\frac{1}{\theta'^H} - 1\right) \cdot \theta^{2H-2} H^2
\end{aligned}
$$

9 Survey

Please estimate, in minutes, for each problem, how much time you spent (a) writing code and (b) waiting for the results. This will help us calibrate the difficulty for future homeworks.

- **Policy Gradients:** 3, 0
- **Neural Network Baseline:** 1, 2
- **Generalized Advantage Estimation:** 1, 2
- **Hyperparameters and Sample Efficiency:** 1, 26
- **Humanoid:** 0, 18
- **Analysis – applying policy gradients:** 2, 0
- **Analysis – PG variance:** 2, 0
- **Analysis – return-to-go:** 2, 0
- **Analysis – importance sampling:** 2, 0