Assignment 2: Policy Gradients

Students: Jules Merigot, Thomas Boudras

Due February 14, 2024

4 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.)

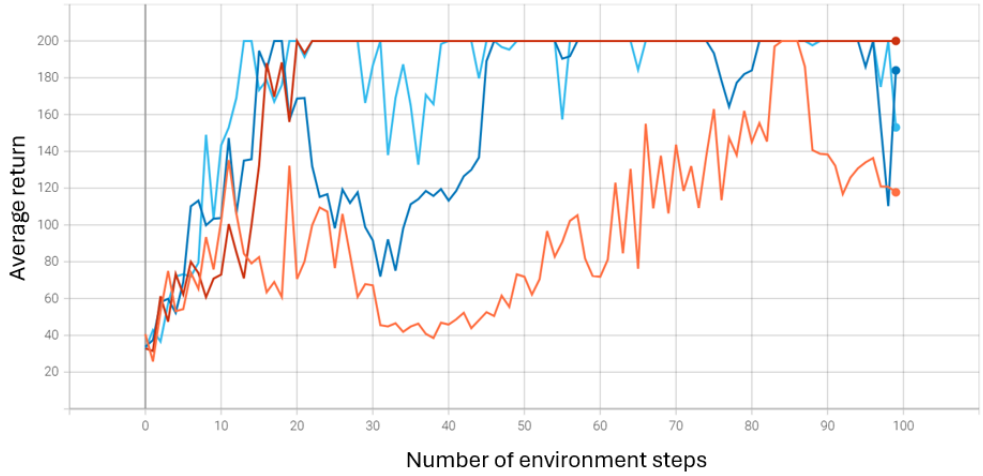


Figure 1: Learning curve for the small batch experiments
Legend: cartpole, cartpole_na, cartpole_rtg, cartpole_rtg_na

  - In the second graph, compare the learning curves for the experiments prefixed with `cartpole_lb`. (The large batch experiments.)

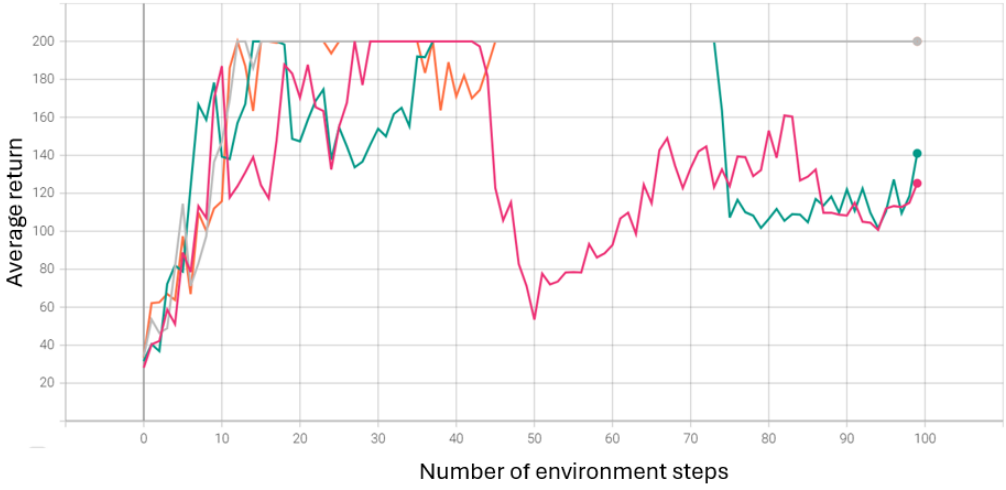


Figure 2: Learning curve for the large batch experiments
Legend: cartpole_lb, cartpole_lb_na, cartpole_lb_rtg, cartpole_lb_rtg_na

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?**

    We can see that using reward-to-go is practically better everywhere, and the value of the average return is higher almost everywhere.

  - **Did advantage normalization help?**

    The average return, and thus the model, is systematically better when batch normalization is used, converging much faster towards the maximum of 200.

  - **Did the batch size make an impact?**

    We can see that going from a batch size of 1000 to 4000 allows faster convergence towards the maximum value in some cases. However, in cases where normalization advantages are used, for example, the difference is not so significant.

  - **Provide the exact command line configurations (or `#@params` settings in Colab) you used to run your experiments, including any parameters changed from their defaults.**

    We have used exactly the same commands as those given in the statement :

```
python cs285/scripts/run_hw2.py --env\_name CartPole-v0 -n 100 -b 1000 --exp\
    _name cartpole
python cs285/scripts/run_hw2.py --env\_name CartPole-v0 -n 100 -b 1000 -rtg --
    exp\_name cartpole\_rtg
python cs285/scripts/run_hw2.py --env\_name CartPole-v0 -n 100 -b 1000 -na --
    exp\_name cartpole\_na
python cs285/scripts/run_hw2.py --env\_name CartPole-v0 -n 100 -b 1000 -rtg -
    na --exp\_name cartpole\_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
```

5 Neural Network Baseline

- Plot a learning curve for the baseline loss.

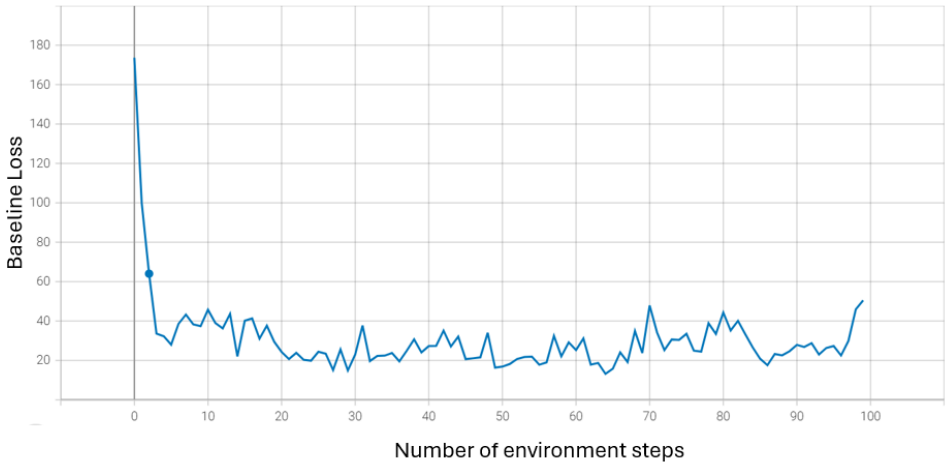


Figure 3: Learning curve for the baseline loss

- Plot a learning curve for the eval return. You should expect to achieve an average return over 300 for the baselined version.

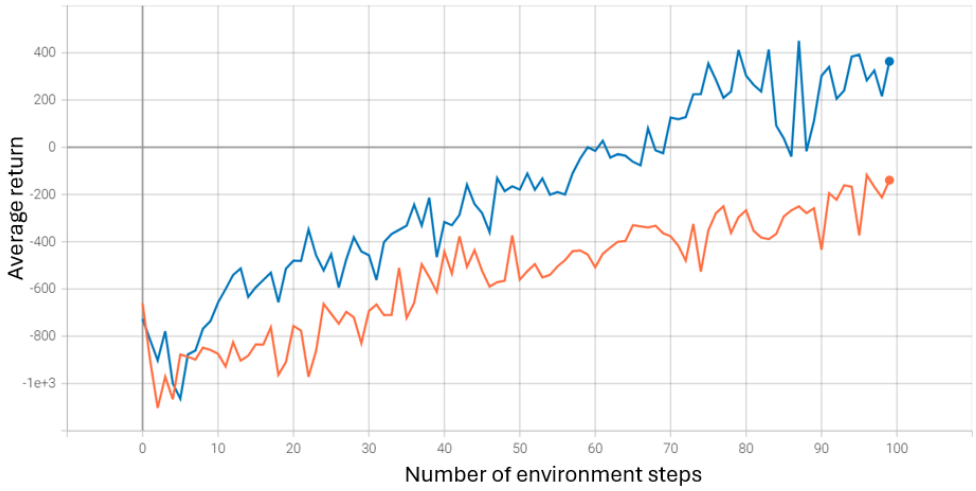


Figure 4: Learning curve to see the impact of baseline
Legend: without baseline, with baseline

- 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?

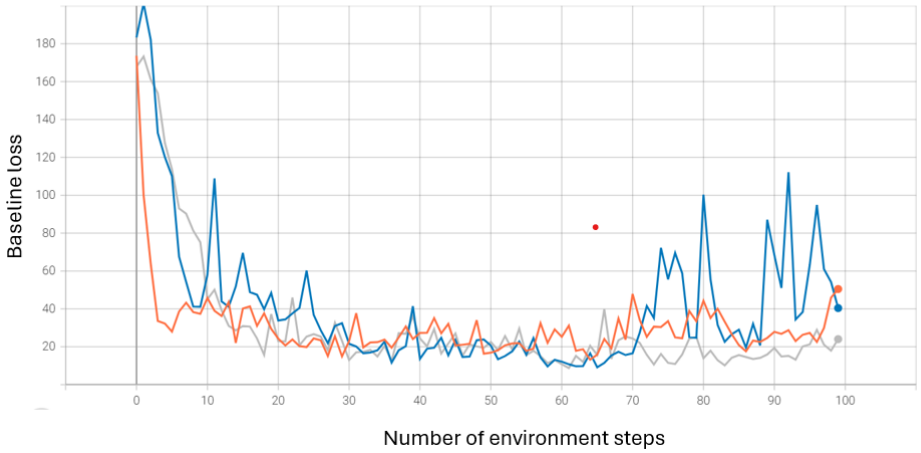


Figure 5: Learning curve to see the impact of baseline gradient steps and baseline learning rate
Legend: $bgs = 5, blr = 0.01$, $bgs = 1, blr = 0.01$, $bgs = 5, blr = 0.001$

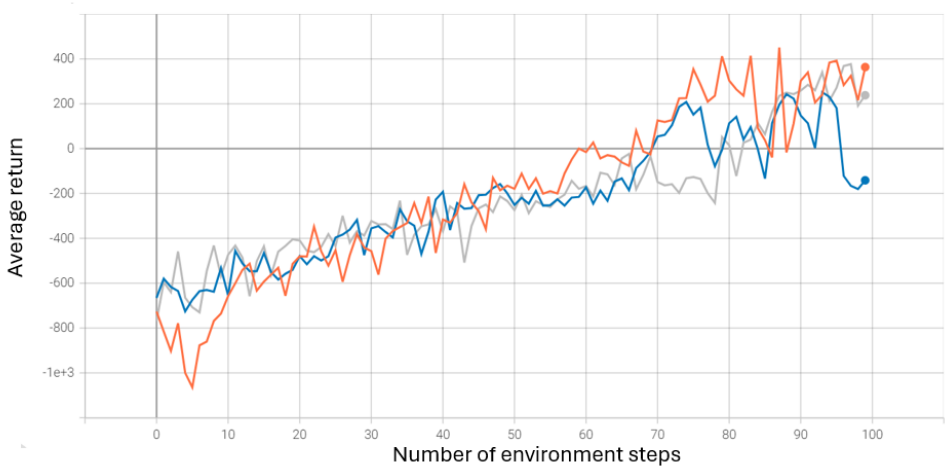


Figure 6: Learning curve to see the impact of baseline gradient steps and baseline learning rate
Legend: $bgs = 5, blr = 0.01$, $bgs = 1, blr = 0.01$, $bgs = 5, blr = 0.001$

There is no really significant difference when these two values are lowered. We can simply see that convergence is a little slower, but only by a relatively small amount. In addition, we are a little less stable by reducing the gradient step, but this too is slight.

- **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!

6 Generalized Advantage Estimation

This part can't be done because we didn't manage to import the box2d package despite several attempts (try on several computers, try on other version of python ...)

- 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.

7 Hyperparameter Tuning

1. Provide a set of hyperparameters that achieve high return on `InvertedPendulum-v4` in as few environment steps as possible.
   - discount factor : 1
   - Network size : 4 layers of neurons of size 64
   - Batch size : 1000
   - Learning rate 5×10^{-3}
   - Use return-to-go : yes
   - Normalization of advantages : yes
   - Use GAE : Yes with $\lambda = 0.9$
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.

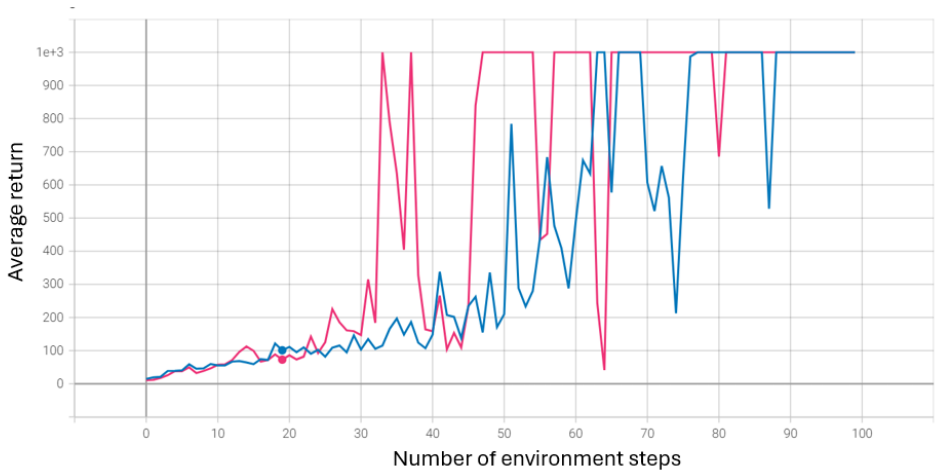


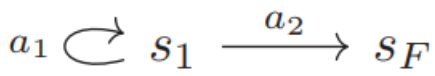
Figure 7: Learning curve to see the impact of baseline
Legend: With hyperparameters, Default settings

8 (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).

9 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:

   To compute the gradient of the expected return $J(\theta)$ with respect to the parameter θ using policy gradients, we can start by considering the definition of $J(\theta) = \mathbb{E}_{\pi_\theta} R(\tau)$, where $R(\tau)$ is the total reward obtained from the trajectory τ. Since we are not using discounting, the total reward $R(\tau)$ will be the sum of rewards obtained at each time step until the termination of the episode.

   In this case, the only way to accumulate reward is by taking action $a1$ in state $s1$, which gives a reward of 1 and keeps the agent in $s1$. Taking action $a2$ results in termination with a reward of 0. Hence, the expected return can be written as:

   $$J(\theta) = \sum_{k=1}^{\infty} \theta^{k-1} = \frac{1}{1-\theta}$$

   The hint provided tells us how to compute the sum of a geometric series when taking the derivative with respect to α. We can apply this to our problem by substituting α with θ:

   $$\frac{d}{d\theta}\sum_{k=1}^{\infty}\theta^{k-1} = \sum_{k=1}^{\infty}\frac{d}{d\theta}\theta^{k-1} = \sum_{k=1}^{\infty}(k-1)\theta^{k-2}$$

   Now we can compute the gradient of $J(\theta)$:

   $$\nabla_\theta J(\theta) = \frac{d}{d\theta}J(\theta) = \sum_{k=1}^{\infty}(k-1)\theta^{k-2}$$

   This is a series representation of the derivative of $J(\theta)$. This series can be summed up as a geometric series derivative, and yields the gradient $\nabla_\theta J(\theta)$.

   Since the sum of the series has a known closed form, we can differentiate this closed form directly with respect to θ to find the derivative of the sum of the series, which represents the policy gradient.

This approach is valid due to the uniform convergence of the power series for $|\theta| < 1$, which allows us to interchange the sum and the derivative operations. In other words, under the conditions that guarantee the convergence of the series, the derivative of the sum is the sum of the derivatives.

We already know this closed form of the sum, which is $\frac{1}{1-\theta}$, which then allows us to take its derivative directly to get the gradient:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \sum_{k=1}^{\infty}(k-1)\theta^{k-2} \\
&= \frac{d}{d\theta}(\frac{1}{1-\theta}) \\
&= \frac{(1)(1-\theta)-(\theta)(-1)}{(1-\theta)^2} \quad \textit{(using the quotient rule: } (\frac{u}{v})' = \frac{u'v-uv'}{v^2}\textit{)} \\
&= \frac{1-\theta+\theta}{(1-\theta)^2} \\
&= \frac{1}{(1-\theta)^2}
\end{aligned}$$

Therefore, the gradient of the expected return with respect to the parameter θ is:

$$\nabla_\theta J(\theta) = \frac{1}{(1-\theta)^2}$$

(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:

The expected return of the policy $J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}R(\tau)$, can be directly computed as the probability of taking action a_1 exactly $k-1$ times, multiplied by the reward received for that action, multiplied by the probability of terminating the process by taking action a_2, summed over an infinite horizon. This can be written as :

$$\begin{aligned}
J(\theta) &= \sum_{k=1}^{\infty}\left(\theta^{k-1}(k-1)(1-\theta)\right) \\
&= (1-\theta)\left(\sum_{k=1}^{\infty}k\theta^{k-1} - \sum_{k=1}^{\infty}\theta^{k-1}\right) \\
&= (1-\theta)\left(\frac{d}{d\theta}\sum_{k=1}^{\infty}\theta^{k} - \sum_{k=0}^{\infty}\theta^{k}\right) \quad \textit{(following the given hint)} \\
&= (1-\theta)\left(\frac{d}{d\theta}\frac{\theta}{1-\theta} - \frac{1}{1-\theta}\right) \\
&= (1-\theta)\left(\frac{1}{(1-\theta)^2} - \frac{1}{1-\theta}\right) \\
&= \frac{\theta}{1-\theta}
\end{aligned}$$

Now, computing the gradient of this expression with respect to θ gives :

$$\nabla_\theta J(\theta) = \frac{d}{d\theta}\left(\frac{\theta}{1-\theta}\right) = \frac{1}{(1-\theta)^2}$$

This result matches the policy gradient that we previously computed in question 1.(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: ***(Not quite sure about this answer, using intuition to solve...)***

   To compute the variance of the policy gradient in closed form, we need to establish the expression for the variance based on the definition provided earlier and then express it in terms of polynomials $P(\theta)$ and $Q(\theta)$.

   Given the policy gradient is $\nabla_\theta J(\theta) = \frac{1}{(1-\theta)^2}$, the square of the policy gradient, which is needed to compute the variance, is:

$$(\nabla_\theta J(\theta))^2 = \left(\frac{1}{(1-\theta)^2}\right)^2 = \frac{1}{(1-\theta)^4}$$

   The variance of a random variable X is given by $Var(X) = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$. Here, X is the policy gradient $\nabla_\theta J(\theta)$, so we need to compute $\mathbb{E}\left[\nabla_\theta J(\theta)^2\right]$ and $(\mathbb{E}\left[\nabla_\theta J(\theta)\right])^2$.

$$\begin{aligned} Var\left(\nabla_\theta J(\theta)\right) &= \mathbb{E}\left[\nabla_\theta J(\theta)^2\right] - \left(\mathbb{E}\left[\nabla_\theta J(\theta)\right]\right)^2 \\ &= \frac{1}{(1-\theta)^4} - \left(\frac{1}{(1-\theta)^2}\right)^2 \\ &= 0 \end{aligned}$$

   The result we obtain is a variance result of 0, which does not seem right. This is because $\nabla_\theta J(\theta)$ is a deterministic function of θ and not a random variable in the traditional sense. Therefore, we interpret $\mathbb{E}[X]$ as the function value itself. This means the variance we're discussing is more about the sensitivity of $\nabla_\theta J(\theta)$ to changes in θ rather than randomness.

   Thus, the computation for understanding how the policy gradient's sensitivity changes with θ will not involve a variance calculation because θ is not random in the context of calculating a policy gradient.

   The simplified expression for the variance of the policy gradient can be interpreted as the square of the policy gradient due to the deterministic nature of θ, and is written as:

$$Var\left(\nabla_\theta J(\theta)\right) = \frac{1}{(1-\theta)^4}$$

   By examining the expression above for its behavior with respect to θ, it's clear that as θ approaches 1, the expression grows infinitely, indicating higher sensitivity or variance in the policy gradient. On the other hand, as θ approaches 0, the expression's value approaches 1, indicating minimal variance.

   Eyeballing the values of θ for which the variance is minimal and maximal:

   - **Minimal variance:** The expression is minimal when θ is close to 0, meaning the sensitivity of the policy gradient to changes in θ is lowest when θ is near 0. However, this interpretation is not perfect because the expression doesn't truly reach a minimum value in the interval of interest $0 \leq \theta < 1$ as it's inversely related to θ.
   - **Maximal variance:** The expression approaches ∞ as θ approaches 1, indicating that the sensitivity of the policy gradient to changes in θ is highest near $\theta = 1$. This is due to the denominator of the variance approaching zero, which reflects a steep increase in the policy gradient's magnitude with small changes in θ near this value.

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

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

   Solution:

   The modified policy gradient, when incorporating the return-to-go as an advantage estimator, is formulated as follows:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) \cdot A(a,s)]$$

   In this context, $A(a,s)$ represents the advantage function, defined as the difference between the return-to-go after taking action a in state s and the average return from state s, denoted by $\overline{R}_s$. Specifically :

$$A(a,s) = R_{to-go}(a,s) - \overline{R}_s$$

   where $R_{to-go}(a,s)$ is the return-to-go after taking action a in state s.

   To demonstrate that the advantage estimator is unbiased, we examine its expectation:

$$\begin{aligned}
\mathbb{E}_{a,s\sim\pi_\theta}[A(a,s)] &= \mathbb{E}_{a,s\sim\pi_\theta}[R_{to-go}(a,s) - \overline{R}_s] \\
&= \mathbb{E}_{a,s\sim\pi_\theta}[R_{to-go}(a,s)] - \mathbb{E}_{a,s\sim\pi_\theta}[\overline{R}_s] \\
&= \overline{R}_s - \overline{R}_s \\
&= 0
\end{aligned}$$

   This calculation shows that the expected value of the advantage function, when averaged over all actions and states according to the policy π_θ, equals zero. This property confirms that the modified policy gradient, when incorporating the return-to-go as an advantage estimator, is unbiased. An unbiased advantage estimator ensures that, on average, the estimator does not systematically overestimate or underestimate the true advantage, thereby allowing for an effective and accurate update of the policy parameters θ in the direction of improving the expected return $J(\theta)$.

   Therefore, the modified policy gradient incorporating return-to-go as an advantage estimator is unbiased.

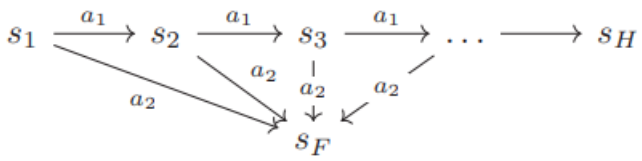
   (b) Compute the variance of the return-to-go policy gradient and plot it on $[0,1]$ alongside the variance of the original estimator.

   Solution:

   We were not really sure how to calculate or explain this. However, our intuition points to the fact that the variance would be zero for $0 \leq \theta < 1$, and would increase towards ∞ as θ approaches 1. This is similar to our answer to question 2.

   The variance of the return-to-go policy would be very similar to the variance of the original estimator, but would still have a variance of zero when θ approaches 1.

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:

Given a finite-horizon H-step MDP with sparse reward, we construct the policy gradient with importance sampling to evaluate the policy π_θ using trajectories generated from a different policy $\pi_{\theta'}$, known as the behavior policy, given by:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_{\theta'}}\left[\frac{\pi_{\theta'}(\tau)}{\pi_\theta(\tau)} R(\tau)\nabla_\theta \log \pi_\theta(\tau)\right]$$

We can then use the likelihood ratio trick to rewrite the above expression. The trajectory τ consists of states and actions $(s_1, a_1, s_2, a_2, ..., s_H, a_H)$. The importance sampling ratio $\rho(\tau)$ is given by the product of the ratios of the probabilities of each action taken under the policies at each step in the trajectory:

$$\rho(\tau) = \frac{\prod_{t=1}^{H} \pi_\theta(a_t|s_t)}{\prod_{t=1}^{H} \pi_{\theta'}(a_t|s_t)}$$

Therefore, the policy gradient using importance sampling can be written as:

$$\begin{aligned} \nabla_\theta J(\theta) &= \mathbb{E}_{\tau\sim\pi_{\theta'}}\left[\frac{\pi_{\theta'}(\tau)}{\pi_\theta(\tau)} R(\tau)\nabla_\theta \log \pi_\theta(\tau)\right] \\ &= \mathbb{E}_{\tau\sim\pi_{\theta'}}\left[\rho(\tau) R(\tau) \sum_{t=1}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t)\right] \end{aligned}$$

Here, $R(\tau)$ is the reward for the entire trajectory, which is 1 if it ends at s_H and 0 otherwise. The expectation is over trajectories sampled from the behavior policy $\pi_{\theta'}$, and $\nabla_\theta \log \pi_\theta(a_t|s_t)$ is the gradient of the log-probability of taking action a_t in state s_t under the target policy.

(b) Compute its variance. How does it change when H becomes large?

Solution:

The variance of the policy gradient using importance sampling is a measure of how much the gradient estimates can vary when computed from samples of trajectories. A high variance is problematic

because it can lead to unstable policy updates. To compute this variance, we apply the law of total variance applied as follows:

$$\begin{aligned}\mathrm{Var}(\nabla_\theta J(\theta)) &= \mathbb{E}_{\tau\sim\pi_{\theta'}}\left[\left(\rho(\tau)R(\tau)\nabla_\theta \log \pi_\theta(\tau)\right)^2\right] - \left(\mathbb{E}_{\tau\sim\pi_{\theta'}}\left[\rho(\tau)R(\tau)\nabla_\theta \log \pi_\theta(\tau)\right]\right)^2 \\ &= \mathbb{E}_{\tau\sim\pi_{\theta'}}\left[\rho(\tau)^2 R(\tau)^2 \left(\nabla_\theta \log \pi_\theta(\tau)\right)^2\right] - \left(\nabla_\theta J(\theta)\right)^2\end{aligned}$$

Here, the first term represents the expected square of the weighted gradient, and the second term is the square of the expected weighted gradient. Subtracting the square of the expectation from the expectation of the square provides the variance. The expectation is computed under the behavior policy $\pi_{\theta'}$ since this is the policy from which our data is sampled.

When the horizon H becomes large, the importance sampling ratio $\rho(\tau)$ squared may lead to high variance in the policy gradient. This is because $\rho(\tau)$ is a product of H terms, each of which can contribute multiplicatively to the variance.

10 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:** 360 minutes, mainly the time to complete the code and then debug it
- **Neural Network Baseline:** 200 minutes, mainly the time to complete the code and then debug it
- **Generalized Advantage Estimation:** 150 minutes without being able to run the package.
- **Hyperparameters and Sample Efficiency:** 120 minutes mainly taken up by the calculation time, which was relatively long for the number of trials we did.
- **Analysis – applying policy gradients:** 3 hours for (a), 3 hours for (b) (a lot of rewriting and self-correcting, still not confident on the answer for part (a));
- **Analysis – PG variance:** 4 hours (mainly intuitive answer, not very confident) ;
- **Analysis – return-to-go:** 1 hour for (a), 1 hour for (b) but not a solid result;
- **Analysis – importance sampling:** 2 hours for (a), 1 hour for (b) (also not a solid result, mostly intuitive explanation)