CAS4160 Homework 2: Policy Gradients

[JongWook Jeon]
[2020142139]

1 Introduction

The goal of this assignment is to help you understand **Policy Gradient Methods**, including variance reduction tricks, such as reward-to-go, neural network baselines, and Generalized Advantage Estimator (GAE). Finally, you will implement Proximal Policy Optimization (PPO), a widely used on-policy RL algorithm that works very well in practice. Here is the link to this homework report template in Overleaf.

2 Review

2.1 Policy Gradient

Recall that the reinforcement learning objective is to learn θ^* that maximizes the objective function:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[r(\tau)], \tag{1}$$

where each rollout τ is of length T, as follows:

$$\pi_\theta(\tau) = p\left(\mathbf{s}_0, \mathbf{a}_0, \ldots, \mathbf{s}_{T-1}, \mathbf{a}_{T-1}, \mathbf{s}_T\right) = p\left(\mathbf{s}_0\right) \prod_{t=0}^{T-1} \pi_\theta\left(\mathbf{a}_t \mid \mathbf{s}_t\right) p\left(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t\right),$$

and

$$r(\tau) = r\left(\mathbf{s}_0, \mathbf{a}_0, \ldots, \mathbf{s}_{T-1}, \mathbf{a}_{T-1}\right) = \sum_{t=0}^{T-1} r\left(\mathbf{s}_t, \mathbf{a}_t\right).$$

Note: In this assignment, t starts from 0, not from 1, which is easier to implement.

The policy gradient approach is to directly take the gradient of this objective:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[r(\tau)] \\
&= \nabla_\theta \int \pi_\theta(\tau) r(\tau) d\tau \\
&= \int \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) r(\tau) d\tau \qquad (\because \nabla_\theta \pi_\theta(\tau) = \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau)) \\
&= \mathbb{E}_{\tau \sim \pi_\theta(\tau)}\left[\nabla_\theta \log \pi_\theta(\tau) r(\tau)\right].
\end{aligned}$$

In practice, the expectation over trajectories τ can be approximated from a batch of N sampled trajectories:

$$\begin{aligned}
\nabla_\theta J(\theta) &\approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log \pi_\theta\left(\tau_i\right) r\left(\tau_i\right) \\
&= \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta\left(\mathbf{a}_{it} \mid \mathbf{s}_{it}\right)\right) \left(\sum_{t=0}^{T-1} r\left(\mathbf{s}_{it}, \mathbf{a}_{it}\right)\right).
\end{aligned} \tag{2}$$

Here, we see that the policy π_θ is a probability distribution over the action space, conditioned on the state. In the agent-environment loop, the agent samples an action $\mathbf{a}_t$ from $\pi_\theta(\cdot \mid \mathbf{s}_t)$ and the environment responds with a reward $r(\mathbf{s}_t, \mathbf{a}_t)$.

2.2 Variance Reduction

2.2.1 Reward-to-go

One way to reduce the variance of the policy gradient is to exploit causality: the notion that the policy cannot affect the rewards in the past. This yields the following modified objective, where the sum of rewards here does not include the rewards achieved prior to the time step at which the policy is being queried. This sum of rewards is a sample estimate of the Q function, and is referred to as the "reward-to-go":

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(\mathbf{a}_{it} \mid \mathbf{s}_{it}) \underbrace{\left(\sum_{t'=t}^{T-1} r(\mathbf{s}_{it'}, \mathbf{a}_{it'}) \right)}_{\text{reward-to-go}}. \tag{3}$$

2.2.2 Discounting

Multiplying a discount factor γ to the rewards can be interpreted as encouraging the agent to focus more on the rewards that are closer in time, and less on the rewards that are further in the future. This can also be thought of as a means for reducing variance as there can be more variance and uncertainty when considering faraway futures.

We can apply the discount on the rewards from full trajectory in Equation (2):

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(\mathbf{a}_{it} \mid \mathbf{s}_{it}) \right) \left(\sum_{t'=0}^{T-1} \gamma^{t'-1} r(\mathbf{s}_{it'}, \mathbf{a}_{it'}) \right). \tag{4}$$

We can also apply the discount on the "reward-to-go" in Equation (3):

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(\mathbf{a}_{it} \mid \mathbf{s}_{it}) \left(\sum_{t'=t}^{T-1} \gamma^{t'-t} r(\mathbf{s}_{it'}, \mathbf{a}_{it'}) \right). \tag{5}$$

2.2.3 Baseline

Another variance reduction method is to subtract a baseline (that is only conditioned on the current state $\mathbf{s}_t$) from the sum of rewards:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) [r(\tau) - b(\mathbf{s}_t)] \right]. \tag{6}$$

This leaves the policy gradient unbiased because

$$\begin{aligned} \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) b(\mathbf{s}_t)] &= \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t)] \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[b(\mathbf{s}_t)] \\ &= 0 \cdot \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[b(\mathbf{s}_t)] = 0. \end{aligned}$$

In this assignment, we will implement a value function $V_\phi^\pi(\mathbf{s}_t)$, which acts as a state-dependent baseline. This value function will be trained to approximate the sum of future rewards starting from a particular state:

$$V_\phi^\pi(\mathbf{s}_t) \approx \sum_{t'=t}^{T-1} \mathbb{E}_{\tau \sim \pi_\theta(\tau)} [r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \mid \mathbf{s}_t]. \tag{7}$$

Then, the approximate policy gradient now looks like this:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta \left(\mathbf{a}_{it} \mid \mathbf{s}_{it} \right) \left(\underbrace{\left(\sum_{t'=t}^{T-1} \gamma^{t'-t} r \left(\mathbf{s}_{it'}, \mathbf{a}_{it'} \right) \right)}_{\text{reward-to-go}} - \underbrace{V_\phi^\pi \left(\mathbf{s}_{it} \right)}_{\text{baseline}} \right). \tag{8}$$

2.2.4 Generalized Advantage Estimator (GAE)

The quantity $\left(\sum_{t'=t}^{T-1} \gamma^{t'-t} r \left(\mathbf{s}_{t'}, \mathbf{a}_{t'} \right) \right) - V_\phi^\pi \left(\mathbf{s}_t \right)$ from the previous policy gradient expression (removing the index i for clarity) can be interpreted as an estimate of the advantage function:

$$A^\pi \left(\mathbf{s}_t, \mathbf{a}_t \right) = Q^\pi \left(\mathbf{s}_t, \mathbf{a}_t \right) - V^\pi \left(\mathbf{s}_t \right), \tag{9}$$

where $Q^\pi \left(\mathbf{s}_t, \mathbf{a}_t \right)$ is estimated using Monte Carlo returns and $V^\pi \left(\mathbf{s}_t \right)$ is estimated using the learned value function V_ϕ^π. We can further reduce variance by also using V_ϕ^π in place of the Monte Carlo returns to estimate the advantage function as:

$$A^\pi \left(\mathbf{s}_t, \mathbf{a}_t \right) \approx \delta_t = r \left(\mathbf{s}_t, \mathbf{a}_t \right) + \gamma V_\phi^\pi \left(\mathbf{s}_{t+1} \right) - V_\phi^\pi \left(\mathbf{s}_t \right), \tag{10}$$

with the edge case $\delta_{T-1} = r \left(\mathbf{s}_{T-1}, \mathbf{a}_{T-1} \right) - V_\phi^\pi \left(\mathbf{s}_{T-1} \right)$. However, this comes at the cost of introducing bias to our policy gradient estimate, due to modeling errors in V_ϕ^π. We can instead use a combination of n-step Monte Carlo returns and V_ϕ^π to estimate the advantage function as:

$$A_n^\pi \left(\mathbf{s}_t, \mathbf{a}_t \right) = \sum_{t'=t}^{t+n} \gamma^{t'-t} r \left(\mathbf{s}_{t'}, \mathbf{a}_{t'} \right) + \gamma^n V_\phi^\pi \left(\mathbf{s}_{t+n+1} \right) - V_\phi^\pi \left(\mathbf{s}_t \right). \tag{11}$$

Increasing n incorporates the Monte Carlo returns more heavily in the advantage estimate, which lowers bias and increases variance, while decreasing n does the opposite. Note that $n = T - t - 1$ recovers the unbiased but higher variance Monte Carlo advantage estimate used in (13), while $n = 0$ recovers the lower variance but higher bias advantage estimate δ_t.

We can combine multiple n-step advantage estimates as an exponentially weighted average, which is known as the generalized advantage estimator (GAE). Let $\lambda \in [0, 1]$. Then, we define:

$$A_{GAE}^\pi \left(\mathbf{s}_t, \mathbf{a}_t \right) = \frac{1 - \lambda^{T-t-1}}{1 - \lambda} \sum_{n=1}^{T-t-1} \lambda^{n-1} A_n^\pi \left(\mathbf{s}_t, \mathbf{a}_t \right), \tag{12}$$

where $\frac{1-\lambda^{T-t-1}}{1-\lambda}$ is a normalizing constant. Note that a higher λ emphasizes advantage estimates with higher values of n, and a lower λ does the opposite. Thus, λ serves as a control for the bias-variance tradeoff, where increasing λ decreases bias and increases variance. In the infinite horizon case ($T = \infty$), we can show:

$$\begin{aligned} A_{GAE}^\pi \left(\mathbf{s}_t, \mathbf{a}_t \right) &= \frac{1}{1 - \lambda} \sum_{n=1}^{\infty} \lambda^{n-1} A_n^\pi \left(\mathbf{s}_t, \mathbf{a}_t \right) \\ &= \sum_{t'=t}^{\infty} (\gamma\lambda)^{t'-t} \delta_{t'}, \end{aligned}$$

where we have omitted the derivation for brevity (see the GAE paper for details). In the finite horizon case, we can write:

$$A_{GAE}^\pi \left(\mathbf{s}_t, \mathbf{a}_t \right) = \sum_{t'=t}^{T-1} (\gamma\lambda)^{t'-t} \delta_{t'}, \tag{13}$$

which serves as a way we can efficiently implement the generalized advantage estimator, since we can recursively compute:

$$A^{\pi}_{GAE}\left(\mathbf{s}_t, \mathbf{a}_t\right) = \delta_t + \gamma\lambda A^{\pi}_{GAE}\left(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}\right) \tag{14}$$

3 Code Structure Overview

The training begins with the script `run_hw2.py`. This script contains the function `run_training_loop`, where the training, evaluation, and logging happens.

The agent, also defined in `run_hw2.py`, is an instance of `PGAgent` in `cas4160/agents/pg_agent.py`. The `PGAgent` class has two main networks as its variables:

- **actor** network: `MLPPolicyPG` in `cas4160/networks/policies.py`, takes as input observations and outputs actions.
- **critic** network: `ValueCritic` in `cas4160/networks/critics.py`, also takes as input observations but outputs value estimates. You will need to implement this in Section 5.

The training iteration begins with sampling trajectories using `sample_trajectories()` defined in `infrastructure/utils.py` and updating the agent via `PGAgent.update()` in `agents/pg_agent.py`.

In this `update()` procedure, `PGAgent` first processes rewards into Q-values (`PGAgent._calculate_q_vals()`) and then, estimates advantages (`PGAgent._estimate_advantage()`). Then, the actor and the critic are updated via `MLPPolicyPG.update()` in `cas4160/networks/policies.py` and `ValueCritic.update()` in `cas4160/networks/critics.py`, respectively. If the `--use_ppo` flag is set, the `MLPPolicyPG.ppo_update()` method is used instead of the standard `MLPPolicyPG.update()` method for the actor. You will work on the `MLPPolicyPG.ppo_update()` method in Section 7.

4 Policy Gradients

4.1 Implementation

You will be implementing two different return estimators within `pg_agent.py`. Note that these differ only by the starting point of the summation.

The first ("Case 1" within `PGAgent._calculate_q_vals()`) uses the discounted cumulative return of the full trajectory and corresponds to the "vanilla" form of the policy gradient (Equation (2)):

$$r\left(\tau_i\right) = \sum_{t'=0}^{T-1} \gamma^{t'} r\left(\mathbf{s}_{it'}, \mathbf{a}_{it'}\right). \tag{15}$$

The second ("Case 2") uses the "reward-to-go" formulation from Equation (3):

$$r\left(\tau_i\right) = \sum_{t'=t}^{T-1} \gamma^{t'-t} r\left(\mathbf{s}_{it'}, \mathbf{a}_{it'}\right). \tag{16}$$

Implement these return estimators as well as the remaining sections marked TODO in the code. For the small-scale experiments, you may skip the parts that are run only if **nn_baseline is True**; we will return to baselines (`MLPPolicyPG.update()` and `PGAgent._estimate_advantage()`) in Section 5.

4.2 Experiments

Experiment 1 (CartPole). Run multiple experiments with the PG algorithm on the discrete `CartPole-v0` environment, using the following commands:

```
python cas4160/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 1000 \
    --exp_name cartpole

python cas4160/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 1000 \
    -rtg --exp_name cartpole_rtg

python cas4160/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 1000 \
    -na --exp_name cartpole_na

python cas4160/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 1000 \
    -rtg -na --exp_name cartpole_rtg_na

python cas4160/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 4000 \
    --exp_name cartpole_lb

python cas4160/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 4000 \
    -rtg --exp_name cartpole_lb_rtg

python cas4160/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 4000 \
    -na --exp_name cartpole_lb_na

python cas4160/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 4000 \
    -rtg -na --exp_name cartpole_lb_rtg_na
```

Here, each argument specifies:

- `-n` : Number of iterations.
- `-b` : Batch size (number of state-action pairs sampled while acting according to the current policy at each iteration).
- -rtg : Flag: if present, sets `reward_to_go=True`. Otherwise, `reward_to_go=False` by default.
- -na : Flag: if present, sets `normalize_advantages=True`, normalizing the advantages to have a mean of zero and standard deviation of one within a batch.
- –exp_name : Name for experiment, which goes into the name for the data logging directory. If your run succeeds, you will be able to find your tensorboard log data in `hw2_starter_code/data/q2_pg_[--exp_name]_[--env_name]_[current_time]/`.

Various other command line arguments will allow you to set batch size, learning rate, network architecture, and more. You can find list of arguments in `cas4160/scripts/run_hw2.py:main()`.

Note: To generate videos of the policy rollouts, add the flag "`--video_log_freq 10`". This will log the evaluation rollout for every 10 iterations. You can watch the video in the "`Images`" tab on tensorboard.

Deliverables for report:

- 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).

Note: 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).

Note: You can use the example helper script (`cas4160/scripts/parse_tensorboard.py`) to parse the data from the tensorboard logs and plot the figure. Here's an example usage that saves the figure as `output_plot.png`:

```
python cas4160/scripts/parse_tensorboard.py \
  --input_log_files data/[your_log_folder_1] data/[your_log_folder_2] \
  --human_readable_names "Vanilla" "Reward to go" \
  --data_key "Eval_AverageReturn" \
  --title "Your Title Here" \
  --x_label_name "Train Environment Steps" \
  --y_label_name "Eval Return" \
  --output_file "output_plot.png"
```

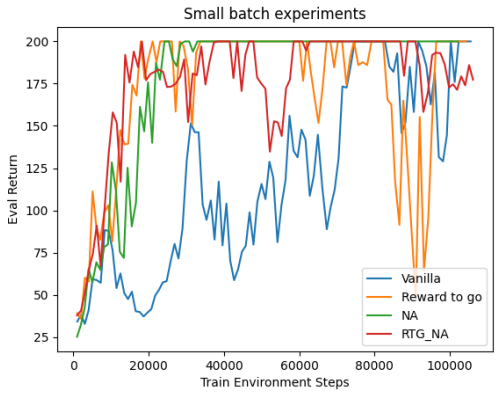
Note that if you add `--plot_mean_std` as the argument, you can plot the standard deviation as a shaded area across the `input_log_files` you specified. You can modify this parsing script for better visualization as you need.

What to Expect:

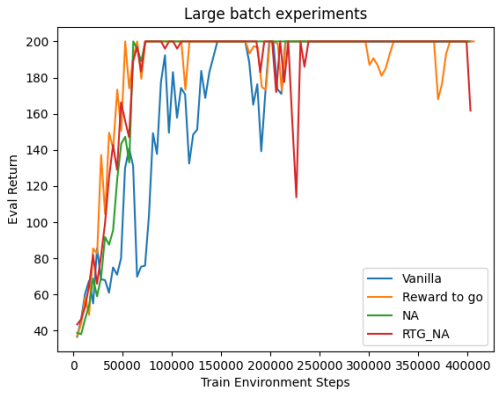
- The best configuration of CartPole in both the large and small batch cases should converge to a maximum score of 200. In addition, each run takes about 5 minutes without video logging.

4.3 Policy Gradient Result

4.3.1 Plot (Eval Average Return with batch size 1000 and 4000):



(a) Batch size 1000



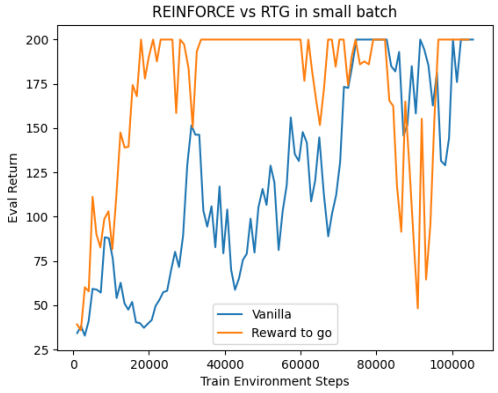
(b) Batch size 4000

Figure 1: Overall caption for both images

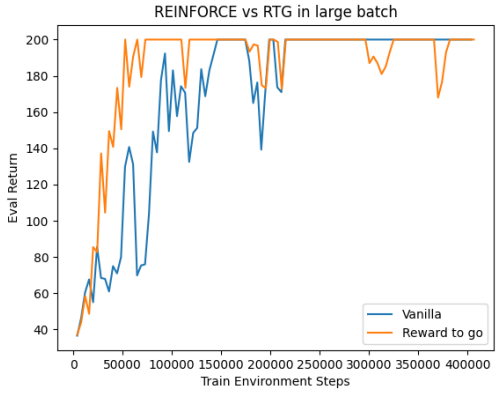
4.3.2 Questions:

Answer the following questions briefly:

- Which value estimator has better performance without advantage normalization: the sum of trajectory rewards (REINFORCE) or the one using reward-to-go?
  Answer:



(a) Batch size 1000



(b) Batch size 4000

The comparison of performance between the REINFORCE method and the reward-to-go method is plotted as above. The Vanilla and the RTG methods have performance increases during train environment steps. At both small and large batch sizes, the RTG method converges to the 200 "Eval Return" more rapidly. This can show that the RTG's performance is better than the REINFORCE method. This is because RTG has a smaller variance by incorporating causality.

Having a smaller variance can lead to better performance. It can be trained to be more stable so that it can converge faster. On the RTG graph, a temporary drop in performance can be observed at the end of the step size. This is because RTG can be attributed to factors such as random seed, learning rate, and exploration. Since performance does not guarantee a monotonically increasing learning curve, at a certain point when updating, it can cause a temporary drop in performance. The Cartport-v0 environment consists of a reward that provides +1 every step. By having a temporary drop, it seems that some factors caused the episode to terminate.

- Did advantage normalization help?
  Answer: **Advantage normalization helps improve performance. Comparing "Vanilla" and "NA" plots, NA shows better performance and converges to the maximum score faster.**
  Advantage normalization also helps with stable training. Without normalization, the gradient could unnecessarily explode and cause instability.
  In the Plot, the RTG+NA has slightly better performance than RTG but the difference in size is barely noticeable. RTG and NA both provide, lower variance while updating. Using both methods can lead to better performance. But In my opinion, I think that the cartpole-v0 environment is too simple so that the performance using both method is similar to using only one.

- Did the batch size make an impact?
  Answer: **Batch size 400 shows better performance compared to batch size 100. This is because having larger batch size, reduces the variance and leads to stable learning. With more batch size, it could learn with more samples and performs better generalization.**
  However, having large batch size can cause increased computational cost and reduced update frequency. Reduced update frequency leads to slower initial training.

- Provide the exact command line configurations you used to run your experiments, including any parameters changed from their defaults.
  Answer:

```
python cas4160/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 1000 \
    --exp_name cartpole

python cas4160/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 1000 \
    -rtg --exp_name cartpole_rtg

python cas4160/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 1000 \
    -na --exp_name cartpole_na

python cas4160/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 1000 \
    -rtg -na --exp_name cartpole_rtg_na

python cas4160/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 4000 \
    --exp_name cartpole_lb

python cas4160/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 4000 \
    -rtg --exp_name cartpole_lb_rtg

python cas4160/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 4000 \
    -na --exp_name cartpole_lb_na

python cas4160/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 4000 \
    -rtg -na --exp_name cartpole_lb_rtg_na
```

The exact command line configurations I used to run my experiments is noted as above. I did not change any parameters and used the given code. the number of iterations are all 100 and the batch size consists of 1000 and 4000. using -rtg presents reward-to-go, -na for normalize.

5 Using a Neural Network Baseline

5.1 Implementation

You will now implement a value function as a state-dependent neural network baseline. This will require filling in some TODO sections skipped in Section 4. In particular:

- This neural network will be trained in the update method of `MLPPolicyPG` along with the policy gradient update.
- In `PGAgent._estimate_advantage()`, the predictions of this network will be subtracted from the reward-to-go to yield an estimate of the advantage. This implements $\left(\sum_{t'=t}^{T-1} \gamma^{t'-t} r\left(\mathbf{s}_{it'}, \mathbf{a}_{it'}\right)\right) - V_\phi^\pi\left(\mathbf{s}_{it}\right)$.
- We will train the baseline network for multiple gradient steps for each policy update, determined by the parameter `baseline_gradient_steps`.

5.2 Experiments

Experiment 2 (HalfCheetah). Next, you will use your baselined policy gradient implementation to learn a controller for `HalfCheetah-v4`.

Run the following commands:

```
# No baseline
python cas4160/scripts/run_hw2.py --env_name HalfCheetah-v4 \
    -n 100 -b 5000 -rtg --discount 0.95 -lr 0.01 \
    --exp_name cheetah
# Baseline
python cas4160/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
```

You might notice that we omitted `-na` (normalize advantages). That's because in reality, advantage normalization is a very powerful trick, and eliminates the need for a baseline in most of the simple environments.

Deliverables:

- Plot a learning curve for the baseline loss.
- Plot a learning curve for the eval return.
- 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?

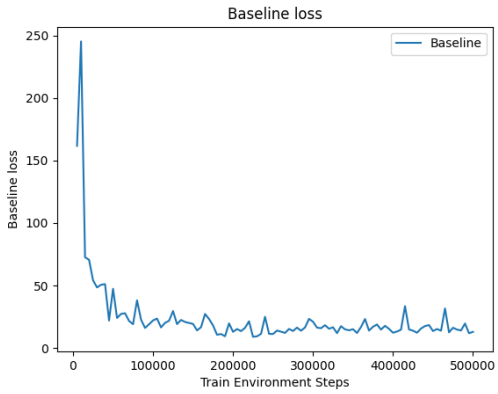
- (Optional) Add `-na` back to see how much it improves things. Also, set `--video_log_freq 10`, then open TensorBoard and go to the "Images" tab to see some videos of your HalfCheetah walking along!

What to Expect:

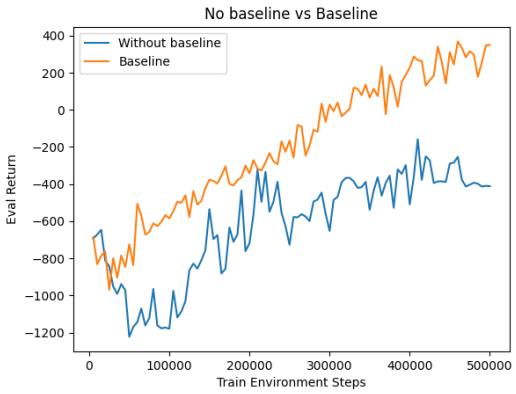
- You should expect to achieve an average return over 300 for the baseline version. In addition, each run takes about 10 minutes without video logging.

5.3 Neural Network Baseline Result

5.3.1 Plot (Baseline Loss, Eval Average Return):



(a) Baseline loss



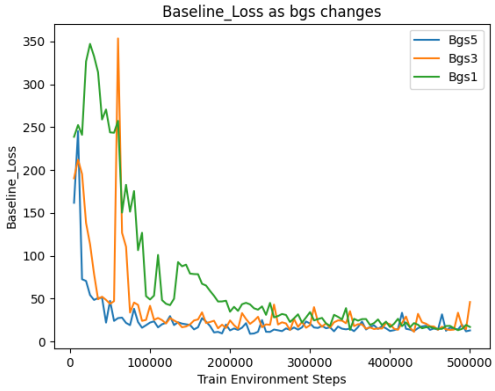
(b) No baseline vs Baseline

Explanation: **Training with a baseline has better performance than without one. The baseline helps reduce variance, leading to more stable training and ultimately better performance. As we can see in the As we can see in the plot, the curve with a baseline shows smaller fluctuations.**
The plot of baseline loss shows that the baseline fits the Q-value estimation well. Since the baseline value function fits the Monte Carlo Q estimation accurately, variance is reduced, resulting in more stable training.

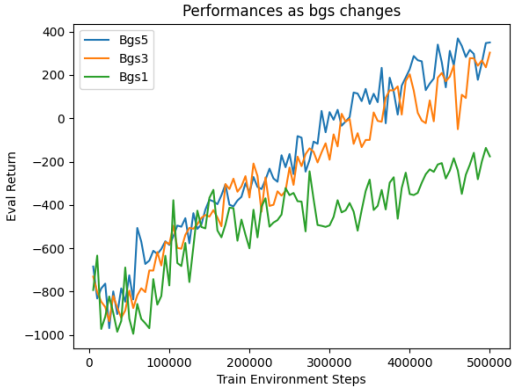
5.3.2 Questions:

Answer the following questions briefly:

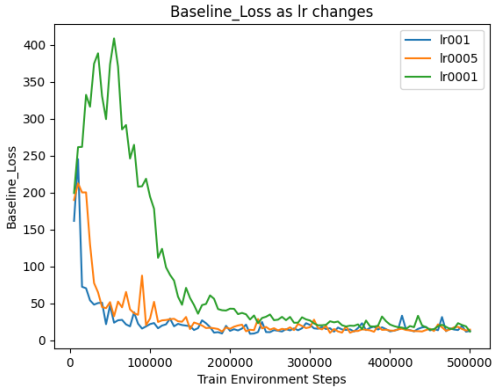
- Run another experiment with a decreased number of baseline gradient steps (`-bgs`) and/or baseline learning rate (`-blr`). How does this affect (a) the baseline learning curve and (b) the performance of the policy?



(a) Baseline loss as Bgs changes



(b) Performance as Bgs changes



(c) Baseline loss as Lr changes

(d) Performance as Lr changes

Answer: **The plot above represents the performance and the baseline loss as Bgs changes and Lr changes. Both changes lead to similar results. First, the baseline-loss. The baseline loss decreases more slowly when either Bgs or Lr becomes smaller. Plots with smaller values of Bgs or Lr converge to zero more slowly and show larger fluctuations. This is because when Bgs is small, the gradient steps for learning the baseline are insufficient, and when Lr is small, updates are too minor. These two factors cause slow and noisy convergence.**
Second, the Performance. The performance drops when Bgs or Lr decreases, and it also becomes more unstable. This happens because smaller Bgs or Lr leads to slower training and higher baseline loss. As a result, the baseline becomes biased, which distorts the advantage function and leads to inaccurate policy gradient updates. Eventually, this lowers the overall performance.
On the other hand, excessively large Bgs and Lr can also make training unstable. Too large Bgs can cause overfitting and large Lr can diverge or oscillate. These factors lead to unstable training. Therefore, it is important to determine appropriate values of Lr and Bgs to fit the baseline well.

6 Generalized Advantage Estimator

6.1 Implementation

You will now use the value function you previously implemented to implement a simplified version of GAE-λ. This will require filling in the remaining TODO section in `PGAgent._estimate_advantage()`.

6.2 Experiments

Experiment 3 (HumanoidStandup-v5). You will now use your implementation of policy gradient with generalized advantage estimation to learn a controller for a version of `HumanoidStandup-v5`. Search over $\lambda \in [0, 0.95, 1]$ to replace $\langle\lambda\rangle$ below. Do not change any of the other hyperparameters (e.g. batch size, learning rate).

```
python cas4160/scripts/run_hw2.py \
    --env_name HumanoidStandup-v5 --ep_len 100 \
    --discount 0.99 -n 50 -l 3 -s 128 -b 2000 -lr 0.001 \
    --use_reward_to_go --use_baseline --gae_lambda <\lambda> \
    --exp_name HumanoidStandup_lambda<\lambda>
```

Deliverables:

- Provide a single plot with the learning curves for the `HumanoidStandup-v5` experiments that you tried. Describe in words how λ affected task performance and training efficiency.
- Consider the parameter λ. What does $\lambda = 0$ correspond to? What about $\lambda = 1$? Relate this to the task performance in `HumanoidStandup-v5` in one or two sentences.

What to Expect:

- The run with the best performance should achieve an average score close to $1.15e4$. In addition, each run takes about 10 minutes without video logging.

6.3 GAE Result

6.3.1 Plot (Eval Average Return with different λ):

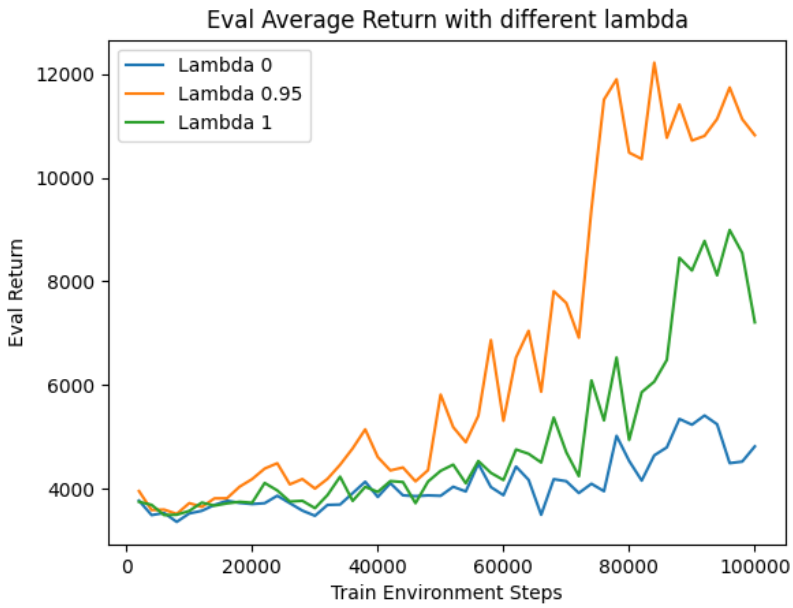


Figure 5: **Eval Average Return with different lambda**

6.3.2 Questions:

Answer the following questions briefly:

- Describe in words how λ affected task performance.
  Answer: **The performance is better in the following order lambda is 0.95, 1 and 0. When lambda is small, close to 0, the GAE is close to a step of TD Advantage. And this has small variance and large bias, which is stable for initial training but can cause low overall performance due to the large bias. When lambda is large, close to 1, the GAE is close to Monte carlo Advantage. And this has high variance but low bias. It can sometimes lead to high performance, but the early training tends to be unstable. When lambda is 0.95 which is between 0 and 1, it could balance the variance-bias trade-off. As a result, the experiment shows that lambda = 0.95 gives the best performance. The result in the plot would provide more reliable results if averaging multiple runs. However, we observed consistent trends across several runs. Therefore we provide this representative plot instead.**

- Consider the parameter λ. What does $\lambda = 0$ correspond to? What about $\lambda = 1$? Relate this to the task performance in `HumanoidStandup-v5` in one or two sentences.
  Answer: **The lambda 0 corresponds to a step TD Advantage. This has low variance but high bias. The lambda 1 corresponds to Monte-carlo Advantage, which has high variance but low bias close to 0.**

7 Proximal Policy Optimization

Finally, you are ready to implement Proximal Policy Optimization (PPO) algorithm. This algorithm uses generalized advantage estimates for calculating its surrogate advantage. There are two primary variants of PPO: PPO-Penalty and PPO-Clip. In this section, we will be implementing PPO-Clip, and its objective is defined by:

$$L\left(\mathbf{s}, \mathbf{a}, \theta_k, \theta\right) = \min\left(\frac{\pi_\theta(\mathbf{a} \mid \mathbf{s})}{\pi_{\theta_k}(\mathbf{a} \mid \mathbf{s})} A^{\pi_{\theta_k}}(\mathbf{s}, \mathbf{a}), \quad \text{clip}\left(\frac{\pi_\theta(\mathbf{a} \mid \mathbf{s})}{\pi_{\theta_k}(\mathbf{a} \mid \mathbf{s})}, 1-\epsilon, 1+\epsilon\right) A^{\pi_{\theta_k}}(\mathbf{s}, \mathbf{a})\right),$$

where θ_k refers to the old policy parameters before the update.

The main idea is that after an update, the new policy should be not too far from the old policy. For that, ppo uses clipping to avoid too large update. Having this form of regulation enables multiple epochs of minibatch updates with the same data, resulting in better stability and sample efficiency. If you want to know more about PPO, here is well-explained documents [OpenAI Spinning Up, PPO paper].

7.1 Implementation

To implement PPO, you need to fill in the TODO section in `MLPPolicyPG.ppo_update()` and `PGAgent.update()`. Please refer to the surrogate objective defined above for calculating the loss.

7.2 Experiments

Experiment 4 (Reacher-v4). Use your implementation of PPO with generalized advantage estimation to learn a controller for `Reacher-v4`. Run the following command to run the experiment.

```
# Reacher (Baseline)
python cas4160/scripts/run_hw2.py \
    --env_name Reacher-v4 --ep_len 1000 \
    --discount 0.99 -n 100 -b 5000 -lr 0.003 \
    -na --use_reward_to_go --use_baseline --gae_lambda 0.97 \
    --exp_name reacher
# Reacher (PPO)
python cas4160/scripts/run_hw2.py \
    --env_name Reacher-v4 --ep_len 1000 \
    --discount 0.99 -n 100 -b 5000 -lr 0.003 \
    -na --use_reward_to_go --use_baseline --gae_lambda 0.97 \
    --use_ppo --n_ppo_epochs 4 --n_ppo_minibatches 4 \
    --exp_name reacher_ppo
```

Deliverables:

- Provide a single plot with the learning curves for the Reacher-v4 experiments that you tried. Compare the performances of the final agent with and without using PPO.
- If we do not use surrogate objective and do multiple batch updates, what would happen? Justify the result.
- (Optional) You can verify the above question by changing "`actor.ppo_update()`" to "`actor.update()`" and comparing the performance.

What to Expect:

- The run with PPO should achieve an average score close to -10 if correctly implemented (at least -15). In addition, each run takes about 14 minutes without video logging.

7.3 PPO Result

7.3.1 Plot (Eval Average Return for PPO and the baseline):

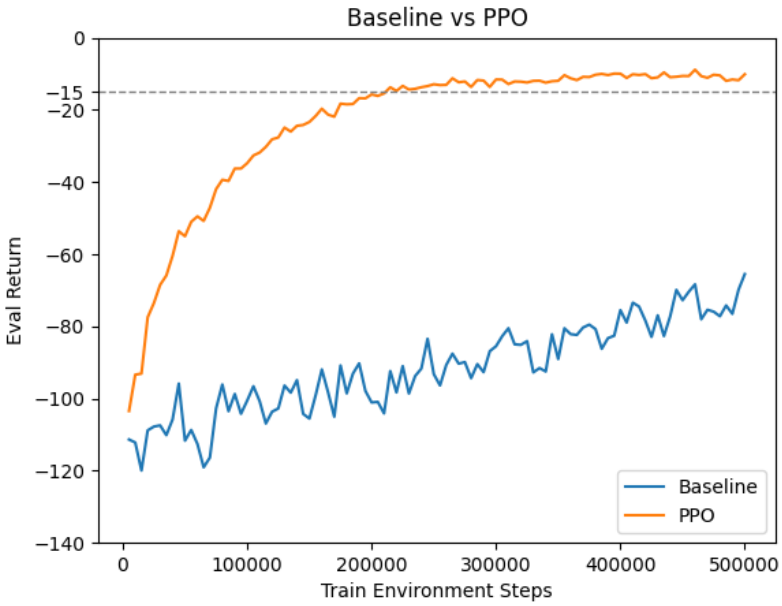


Figure 6: **Baseline vs PPO**

Explanation: **We compare the baseline policy gradient approach to the PPO-based method. The plot shows the average eval return on the y-axis and training steps on the x-axis. Usually, the PPO curve goes up more steadily and can reach around -10, while the baseline might get stuck around -80 or so. This happens because PPO's clipping function helps prevent very large policy changes, which makes learning more stable. The baseline approach can still improve, but it may jump around a lot or fail to converge as well as PPO.**

7.3.2 Questions:

Answer the following questions briefly:

- If we do not use surrogate objective and do multiple batch updates, what would happen? Justify the result.
  Answer: **If we just do many gradient steps on the same batch without surrogate objective, the policy can shift far from the distribution that created that batch. That means the gradient becomes less accurate, and training can get unstable or collapse. The surrogate objective in PPO is designed to keep the updated policy close to the old one, so multiple passes over the same data don't break the assumptions. Without it, we might see sudden drops in performance or inconsistent results.**

8 Discussion

Please provide us a rough estimate, in hours, for each problem, how much time you spent. This will help us calibrate the difficulty for future homework.

- **Policy Gradients**: **05 hours**
- **Neural Network Baseline**: **02 hours**
- **Generalized Advantage Estimator**: **02 hours**
- **Proximal Policy Optimization**: **01 hours**

Feel free to share your feedback here, if any: **We would really appreciate your feedback to improve the reinforcement learning class.**

9 Submission

Please include the code, tensorboard logs data, and the **"report"**. Zip it to `hw2_[YourStudentID].zip`. The structure of the submission file should be:

```
hw2_[YourStudentID].zip
├── hw2_[YourStudentID].pdf
├── cas4160/
│   └── ...codes
├── data/
│   └── ...tensorboard log folders
└── ...
```

Note: Do NOT include the videos (.mp4 files or tensorboard logs) in your submission. Your submission file size should be less than 50MB.