CSCE 790-003, Spring 2022
Assignment 3

Username: [your email username]
Name: [your first and last name]

By turning in this assignment, I agree by the honor code of USC Columbia.

Submission. You need to submit the following files to Blackboard:

- A pdf file named as assignment3_⟨*username*⟩.pdf, where you replace ⟨*username*⟩ with your email username. This pdf file contains your answers to the written problems, including problems 1 and 2.6.
- A zip file named as assignment3_⟨*username*⟩.zip, where you replace ⟨*username*⟩ with your email username. This zip file contains the whole directory of `starter_code_torch` to reproduce your result for problem 2.6.

1 Per-trajectory REINFORCE [25 pt]

In class, the policy gradient expressions we derived are "per-step", that is, the gradient estimator can be calculated for every step t of a sampled trajectory. For example, the REINFORCE estimator is $\nabla_\theta \log \pi_\theta(a_t|s_t) \cdot G_t$. In this problem, we will derive the "per-trajectory" REINFORCE.

Still, we consider a stochastic policy parameterized by θ, $\pi_\theta(\cdot|s)$. The initial state is fixed to be some state s_o. Let $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, ...)$ denote the state-action-reward trajectory. The probability distribution over trajectories under policy π_θ can be expressed as

$$p^{\pi_\theta}(\tau) = \pi_\theta(a_0|s_0)p(s_1|s_0, a_0)\pi_\theta(a_1|s_1)...$$

assuming the reward function is deterministic, $r_t = R(s_t, a_t)$. The discounted total reward of trajectory τ is denoted as

$$G(\tau) := \sum_{t=0}^{H-1} \gamma^t r_t$$

where horizon H can be finite or infinite.

For per-trajectory REINFORCE, we start by decomposing the objective into trajectories, i.e. $V^{\pi_\theta}(s_0) = \sum_\tau [p^{\pi_\theta}(\tau) \cdot G(\tau)]$.

(a) Show that the policy gradient can be expressed as

$$\begin{aligned}\nabla_\theta V^{\pi_\theta}(s_0) =& \nabla_\theta \sum_\tau [p^{\pi_\theta}(\tau) \cdot G(\tau)] \\ =& \mathbb{E}_{\tau \sim p^\pi(\tau)} \left[G(\tau) \cdot \nabla_\theta \log \pi_\theta(\tau) \right].\end{aligned}$$

where $\log \pi_\theta(\tau) := \sum_{t=0}^{H-1} \log \pi_\theta(a_t|s_t)$ Accordingly, the gradient estimator $G(\tau) \cdot \log \pi_\theta(\tau)$ is called the "per-trajectoy" REINFORCE.

HINT:

$$\nabla_\theta \sum_\tau [p^{\pi_\theta}(\tau) \cdot G(\tau)] = \sum_\tau [\nabla_\theta p^{\pi_\theta}(\tau) \cdot G(\tau)] = \sum_\tau [p^{\pi_\theta}(\tau) \nabla_\theta \log p^{\pi_\theta}(\tau) \cdot G(\tau)]$$

$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, ...)$, $p^{\pi_\theta}(\tau) = \pi_\theta(a_0|s_0)p(s_1|s_0, a_0)\pi_\theta(a_1|s_1)..., G(\tau) := \sum_{t=0}^{H-1} \gamma^t R(s_t, a_t)$

$$
\begin{aligned}
V^{\pi_\theta}(s_0) &= \mathbb{E}_\theta[G(s_0)] = \sum_\tau [p^{\pi_\theta}(\tau) \cdot G(\tau)] \\
\nabla_\theta V^{\pi_\theta}(s_0) &= \sum_\tau [\nabla_\theta p^{\pi_\theta}(\tau) \cdot G(\tau)] \\
&= \sum_\tau [p^{\pi_\theta}(\tau) \nabla_\theta [\log p^{\pi_\theta}(\tau)] \cdot G(\tau)] \\
&= \sum_\tau [p^{\pi_\theta}(\tau) \nabla_\theta \log[\pi_\theta(a_0|s_0)p(s_1|s_0, a_0)\pi_\theta(a_1|s_1)...] \cdot G(\tau)] \\
&= \sum_\tau [p^{\pi_\theta}(\tau) \nabla_\theta [\log \pi_\theta(a_0|s_0) + \log p(s_1|s_0, a_0) + \log \pi_\theta(a_1|s_1) + ...] \cdot G(\tau)]
\end{aligned}
$$

As the reward function is deterministic, we have $\log p(s_{t+1}|s_t, a_0) = \log 1 = 0$ at all t:

$$
\begin{aligned}
\nabla_\theta V^{\pi_\theta}(s_0) &= \sum_\tau [p^{\pi_\theta}(\tau) \nabla_\theta [\sum_{t=0}^{H-1} \log \pi_\theta(a_t|s_t)] \cdot G(\tau)] \\
&= \sum_\tau p^{\pi_\theta}(\tau) [\nabla_\theta \log \pi_\theta(\tau) \cdot G(\tau)] \\
&= \mathbb{E}_{\tau \sim p^\pi(\tau)} [G(\tau) \cdot \nabla_\theta \log \pi_\theta(\tau)]
\end{aligned}
$$

(b) A baseline can be introduced to reduce the variance for the "per-trajectoy" REINFORCE. Specifically, we consider using a constant baseline $b \in \mathbb{R}$, which introduces no bias:

$$
\nabla_\theta V^{\pi_\theta}(s_0) = \mathbb{E}_{\tau \sim p^{\pi_\theta}(\tau)} [(G(\tau) - b) \nabla_\theta \log \pi_\theta(\tau)] .
$$

For simplicity, let's consider the 1-d case where $\theta \in \mathbb{R}$. We now look at the variance of the gradient estimator $\widehat{g} := (G(\tau) - b) \nabla_\theta \log \pi_\theta(\tau) \in \mathbb{R}$. What's the optimal baseline that minimizes the variance of $\widehat{g}$?

HINT: Express the variance as $Var[\widehat{g}] = \mathbb{E}[\widehat{g}^2] - (\mathbb{E}[\widehat{g}])^2$. Then view $Var[\widehat{g}]$ as a function of baseline b; find its minimizer b^*.

$$
\begin{aligned}
\mathrm{Var}(\nabla_\theta V^{\pi_\theta}(s_0)) &= \mathrm{Var}\{\mathbb{E}_{\tau \sim p^{\pi_\theta}(\tau)}[(G(\tau) - b) \nabla_\theta \log \pi_\theta(\tau)]\} \\
&= \mathrm{Var}\{\sum_\tau p^{\pi_\theta}(\tau)[(G(\tau) - b) \nabla_\theta \log \pi_\theta(\tau)]\} \\
&= \sum_\tau p^{\pi_\theta}(\tau)^2 \, \mathrm{Var}\{[(G(\tau) - b) \nabla_\theta \log \pi_\theta(\tau)]\}
\end{aligned}
$$

Let $\widehat{g} := (G(\tau) - b)\nabla_\theta \log \pi_\theta(\tau) \in \mathbb{R}$,

$$\mathrm{Var}(\nabla_\theta V^{\pi_\theta}(s_0)) = \sum_\tau p^{\pi_\theta}(\tau)^2 \{\mathbb{E}(\hat{g}^2) - \mathbb{E}(\hat{g})^2\}$$

Let $\frac{d}{db}\{\sum_\tau p^{\pi_\theta}(\tau)^2[\mathbb{E}(\hat{g}^2) - \mathbb{E}(\hat{g})^2]\} = \sum_\tau p^{\pi_\theta}(\tau)^2 \frac{d}{db}\{\mathbb{E}(\hat{g}^2) - \mathbb{E}(\hat{g})^2\} = 0$. As b introduce no bias and is a constant:

$$\sum_\tau p^{\pi_\theta}(\tau) b \nabla_\theta \log \pi_\theta(\tau) = b \sum_\tau p^{\pi_\theta}(\tau) \nabla_\theta \log \pi_\theta(\tau) = 0$$
$$\mathbb{E}(\frac{d}{db}\hat{g}) = \sum_\tau -p^{\pi_\theta}(\tau) \nabla_\theta \log \pi_\theta(\tau) = 0$$

Then we have:

$$\begin{aligned}
\frac{d}{db}\{\mathrm{Var}(\nabla_\theta V^{\pi_\theta}(s_0))\} &= \sum_\tau p^{\pi_\theta}(\tau)^2 \frac{d}{db}\,\mathbb{E}(\hat{g}^2) - \sum_\tau p^{\pi_\theta}(\tau)^2 \frac{d}{db}\,\mathbb{E}(\hat{g})^2 \\
&= \sum_\tau p^{\pi_\theta}(\tau)^2\,\mathbb{E}(\frac{d}{db}\hat{g}^2) - \sum_\tau p^{\pi_\theta}(\tau)^2\,\mathbb{E}(\frac{d}{db}\hat{g})^2 \\
&= \sum_\tau p^{\pi_\theta}(\tau)^2\,\mathbb{E}(2\hat{g}\frac{d\hat{g}}{db}) = 0
\end{aligned}$$

$$-2\sum_\tau p^{\pi_\theta}(\tau)^2 \{[(G(\tau) - b)\nabla_\theta \log \pi_\theta(\tau)]\nabla_\theta \log \pi_\theta(\tau)\} = 0$$

$$-2\sum_\tau p^{\pi_\theta}(\tau)^2 G(\tau)[\nabla_\theta \log \pi_\theta(\tau)]^2 + 2\sum_\tau p^{\pi_\theta}(\tau)^2 b[\nabla_\theta \log \pi_\theta(\tau)]^2 = 0$$

$$b\sum_\tau p^{\pi_\theta}(\tau)^2[\nabla_\theta \log \pi_\theta(\tau)]^2 = \sum_\tau p^{\pi_\theta}(\tau)^2 G(\tau)[\nabla_\theta \log \pi_\theta(\tau)]^2$$

$$b = \frac{\sum_\tau p^{\pi_\theta}(\tau)^2 G(\tau)[\nabla_\theta \log \pi_\theta(\tau)]^2}{\sum_\tau p^{\pi_\theta}(\tau)^2[\nabla_\theta \log \pi_\theta(\tau)]^2}$$

2 Implementing Policy Gradient [75 pt]

This problem is adapted from Emma Brunskill's course. The starter code is in folder `starter_code_torch`.

The goal of this problem is to experiment with policy gradient and its variants, including variance reduction methods. Your goals will be to set up policy gradient for both continuous and discrete environments, and implement a neural network baseline for variance reduction. The framework for the policy gradient algorithm is setup in `main.py`, and everything that you need to implement is in the files `network_utils.py`, `policy.py`, `policy_gradient.py` and `baseline_network.py`. The file has detailed instructions for each implementation task, but an overview of key steps in the algorithm is provided here.

2.1 REINFORCE

Recall the policy gradient theorem (finite horizon with $\gamma = 1$): the gradient of the policy performance, denoted as $\nabla_\theta J(\theta)$, can be expressed as

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_t \nabla_\theta \log \pi_\theta(a_t|s_t) Q^{\pi_\theta}(s_t, a_t) \right],$$

where τ is a trajectory. REINFORCE is a Monte Carlo policy gradient algorithm, so we will be using the sampled returns G_t as unbiased estimates of $Q^{\pi_\theta}(s, a)$. The REINFORCE estimator can be expressed as the gradient of the following objective function (e.g., for automatic differentiation):

$$J(\theta) = \frac{1}{\sum_i T_i} \sum_{i=1}^{|D|} \sum_{t=1}^{T_i} \log(\pi_\theta(a_t^i|s_t^i)) G_t^i$$

where D is the set of all trajectories collected by policy π_θ, and $\tau^i = (s_1^i, a_1^i, r_1^i, s_2^i, \ldots, s_{T_i}^i, a_{T_i}^i, r_{T_i}^i)$ is trajectory i (using 1-based indexing for t).

2.2 Baseline

One difficulty of training with the REINFORCE algorithm is that the Monte Carlo sampled return(s) G_t can have high variance. To reduce variance, we subtract a baseline $b_\phi(s)$ from the estimated returns when computing the policy gradient. A good baseline is the state value function, $V^{\pi_\theta}(s)$, which requires a training update to ϕ to minimize the following mean-squared error loss:

$$L_{\text{MSE}}(\phi) = \frac{1}{\sum_i T_i} \sum_{i=1}^{|D|} \sum_{t=1}^{T_i} (b_\phi(s_t^i) - G_t^i)^2$$

2.3 Advantage Normalization

After subtracting the baseline, we get the following new objective function:

$$J(\theta) = \frac{1}{\sum_i T_i} \sum_{i=1}^{|D|} \sum_{t=1}^{T_i} \log(\pi_\theta(a_t^i|s_t^i)) \hat{A}_t^i$$

where

$$\hat{A}_t^i = G_t^i - b_\phi(s_t^i)$$

A second variance reduction technique is to normalize the computed advantages, $\hat{A}_t^i$, so that they have mean 0 and standard deviation 1. From a theoretical perspective, we can consider centering the advantages to be simply adjusting the advantages by a constant baseline, which does not change the policy gradient. Likewise, rescaling the advantages effectively changes the learning rate by a factor of $1/\sigma$, where σ is the standard deviation of the empirical advantages.

2.4 Functions to Implement

The functions that you need to implement in `network_utils.py`, `policy.py`, `policy_gradient.py`, and `baseline_network.py` are enumerated here. Detailed instructions for each function can be found in the comments in each of these files.

Note: The "batch size" for all the arguments is $\sum_i T_i$ since we already flattened out all the episode observations, actions, and rewards for you.

In `network_utils.py`,

- `build_mlp`

In `policy.py`,

- `BasePolicy.act`
- `CategoricalPolicy.action_distribution`
- `GaussianPolicy.__init__`
- `GaussianPolicy.std`
- `GaussianPolicy.action_distribution`

In `policy_gradient.py`,

- `PolicyGradient.init_policy`
- `PolicyGradient.get_returns`
- `PolicyGradient.normalize_advantage`
- `PolicyGradient.update_policy`

In `baseline_network.py`,

- `BaselineNetwork.__init__`
- `BaselineNetwork.forward`
- `BaselineNetwork.calculate_advantage`
- `BaselineNetwork.update_baseline`

2.5 Testing

We have provided some basic tests to sanity check your implementation. **Please note that the tests are not comprehensive, and passing them does not guarantee a correct implementation**. Use the following command to run the tests:

```
python run_basic_tests.py
```

You can also add additional tests of your own design in `tests/test_basic.py`.

2.6 Getting Plots

The general form for running your policy gradient implementation is as follows:

```
python main.py --env-name ENV --seed SEED --no-baseline
```

if not using a baseline, or

```
python main.py --env-name ENV --seed SEED --baseline
```

if using a baseline. Here `ENV` can be `cartpole`, `pendulum`, or `cheetah`, and `SEED` should be a positive integer.

For an environment, choose 3 random seeds and run the algorithm both without baseline and with baseline. Then plot the results using

```
python plot.py --env-name ENV --seeds SEEDS
```

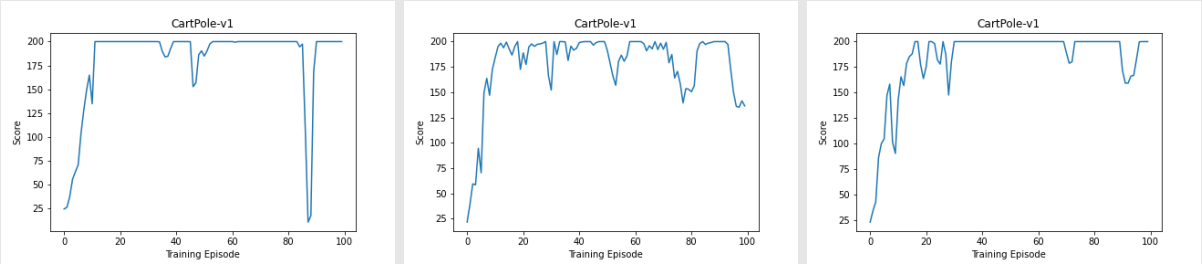
where `SEEDS` should be a comma-separated list of seeds which you want to plot (e.g. `--seeds 1,2,3`).

Since `pendulum` and `cheetah` require mujoco license, you are not required to run them. Even if you run them, exclude their plots from your writeup. Please include the plot for `cartpole` only in your writeup, and comment on whether or not you observe improved performance when using a baseline.

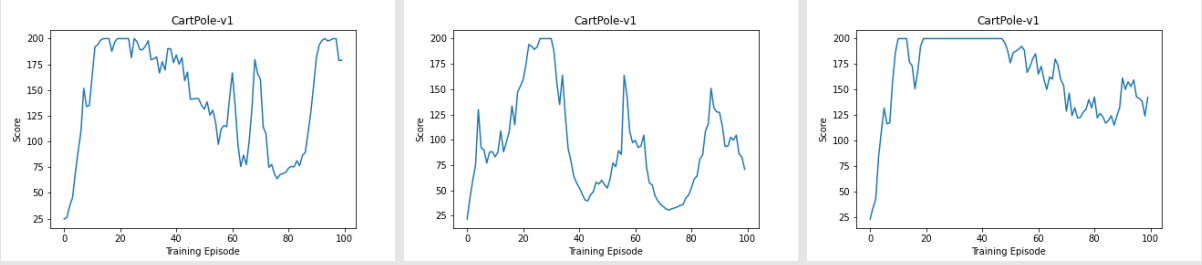
We have the following expectations about performance:

- cartpole: Should reach the max reward of 200 (although it may not stay there)
- pendulum: Should reach the max reward of 1000 (although it may not stay there)
- cheetah: Should reach at least 200 (Could be as large as 950)

Baseline:



No baseline:



3 Regret Analysis of Explore-Then-Commit for MAB [25 pt]

Consider a multi-armed bandit (MAB) specified by

- K arms with $[0,1]$-bounded reward distributions $\nu_1, ..., \nu_K \in \Delta([0,1])$
- Game length T

We use the following notations:

- $\mu_i := \mathbb{E}_{X\sim\nu_i}[X]$, the true mean of arm i
- $i^* := \arg\max_i \mu_i$, the best arm i^*
- $\Delta_i := \mu_{i^*} - \mu_i$, the gap of arm i
- $I_t \in \{1, ..., K\}$ is the arm pulled at round t; $X_t \sim \nu_{I_t}$ is the reward received at round t
- $N_{i,t} := \sum_{s=1}^{t} \mathbb{1}[I_s = i]$, number of pulls of arm i by round t, where $\mathbb{1}[E]$ is the indicator function of event E
- $\hat{\mu}_{i,t} := \frac{1}{N_{i,t}} \sum_{s=1}^{t} X_s \cdot \mathbb{1}[I_s = i]$, empirical mean of arm i by round t

Consider the Explore-Then-Commit algorithm with exploration parameter M:

1. For $t = 1, ..., MK$, pull each arm M times.
2. For $t = MK + 1, ..., T$, pull the empirically best arm $\hat{i} := \arg\max_j \hat{\mu}_{j,MK}$.

Prove that its pseudo-regret is bounded as

$$\overline{R_T} := \sum_i \Delta_i \, \mathbb{E}[N_{i,T}] \leq \sum_i \Delta_i \left(M + 2(T - MK) e^{\frac{-M\Delta_i^2}{2}} \right)$$

Hint: Since

$$\begin{aligned} N_{i,T} &= M + (T - MK) \cdot \mathbb{1}[i = \arg\max_j \hat{\mu}_{j,MK}] \\ \mathbb{E}[N_{i,T}] &= M + (T - MK) \cdot \Pr(i = \arg\max_j \hat{\mu}_{j,MK}) \end{aligned}$$

It suffices to show that for any given suboptimal arm $i : \Delta_i > 0$, $\Pr(i = \arg\max_j \hat{\mu}_{j,MK}) \leq 2e^{\frac{-M\Delta_i^2}{2}}$. We show this next, using $\hat{\mu}_i$ as shorthand for $\hat{\mu}_{i,MK}$.

By Hoeffding's bound, for any given arm j,

$$\begin{aligned} \Pr(\hat{\mu}_j - \mu_j \geq \epsilon) &\leq e^{-2\epsilon^2 M} \\ \Pr(\hat{\mu}_j - \mu_j \leq -\epsilon) &\leq e^{-2\epsilon^2 M} \end{aligned}$$

For any given suboptimal arm $i : \Delta_i > 0$, if we set $\epsilon = \Delta_i/2$ for $j \in \{i, i^*\}$, then arm i will be favored over optimal arm i^* with low probability.

It is obvious that for the operations: $t = 1, ..., MK$, pull each arm M times, then for $t = MK + 1, ..., T$, pull the empirically best arm $\hat{i} := \arg\max_j \hat{\mu}_{j,MK}$:

$$
\begin{aligned}
N_{i,T} =& \sum_{s=1}^{T} \mathbb{1}[I_s = i] = M + (T - MK) \cdot \mathbb{1}[i = \arg\max_j \hat{\mu}_{j,MK}] \\
\mathbb{E}[N_{i,T}] =& M + (T - MK) \cdot \Pr(i = \arg\max_j \hat{\mu}_{j,MK})
\end{aligned}
$$

For any given suboptimal arm $i : \Delta_i > 0$. By Hoeffding's bound, for any given arm j,

$$
\begin{aligned}
\Pr(\hat{\mu}_j - \mu_j \geq \epsilon) &\leq e^{-2\epsilon^2 M} \\
\Pr(\hat{\mu}_j - \mu_j \leq -\epsilon) &\leq e^{-2\epsilon^2 M}
\end{aligned}
$$

For any given suboptimal arm $i : \Delta_i > 0$, if we set $\epsilon = \Delta_i/2$ for $j \in \{i, i^*\}$, then using $\hat{\mu}_i$ as shorthand for $\hat{\mu}_{i,MK}$

$$
\begin{aligned}
&\Pr(i = \arg\max_j \hat{\mu}_{j,MK}) = \Pr(\hat{\mu}_{i,MK} \geq \max_{i \neq j} \hat{\mu}_{j,MK}) \\
\leq& \Pr(\hat{\mu}_{i,MK} \geq \hat{\mu}_{i_*,MK}) = \Pr(\hat{\mu}_{i,MK} - \hat{\mu}_{i_*,MK} + \mu_{i^*} - \mu_i \geq \mu_{i^*} - \mu_i) \\
=& \Pr(\hat{\mu}_{i,MK} - \hat{\mu}_{i_*,MK} - \mu_{i^*} + \mu_i \geq -\mu_{i^*} + \mu_i) \\
\leq& \Pr(-\mu_{i^*} + \mu_i \leq 2\hat{\mu}_i - 2\mu_i \leq \mu_{i^*} - \mu_i) \\
=& \Pr(-\Delta_i/2 \leq \hat{\mu}_i - \mu_i \leq \Delta_i/2) \\
=& \Pr(\hat{\mu}_i - \mu_i \geq \Delta_i/2) + \Pr(\hat{\mu}_i \mu_i \leq -\Delta_i/2) \\
\leq& e^{-2(\Delta_i/2)^2 M} + e^{-2(\Delta_i/2)^2 M} = 2e^{\frac{-M\Delta_i^2}{2}}
\end{aligned}
$$

$$
\begin{aligned}
\overline{R_T} &= \sum_i \Delta_i \, \mathbb{E}[N_{i,T}] \\
&= \sum_i \Delta_i \{M + (T - MK) \cdot \Pr(i = \arg\max_j \hat{\mu}_{j,MK})\} \\
&\leq \sum_i \Delta_i \{M + (T - MK) \cdot 2e^{\frac{-M\Delta_i^2}{2}}\}
\end{aligned}
$$