CS 234: Assignment #3

Due date: February 26, 2021 at 6:00 PM (18:00) PST

These questions require thought, but do not require long answers. Please be as concise as possible.

We encourage students to discuss in groups for assignments. **However, each student must finish the problem set and programming assignment individually, and must turn in her/his assignment.** We ask that you abide by the university Honor Code and that of the Computer Science department, and make sure that all of your submitted work is done by yourself. If you have discussed the problems with others, please include a statement saying who you discussed problems with. Failure to follow these instructions will be reported to the Office of Community Standards. We reserve the right to run a fraud-detection software on your code.

Please review any additional instructions posted on the assignment page at http://web.stanford.edu/class/cs234/assignments.html. When you are ready to submit, please follow the instructions on the course website.

1 Policy Gradient Methods (50 pts coding + 15 pts writeup)

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.

1.1 REINFORCE

Recall the policy gradient theorem,

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s,a)\right]$$

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:

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

where D is the set of all trajectories collected by policy π_θ, and $\tau^i = (s_0^i, a_0^i, r_0^i, s_1^i, \dots, s_{T_i}^i, a_{T_i}^i, r_{T_i}^i)$ is trajectory i.

1.2 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:

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

1.3 Advantage Normalization

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

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

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.

1.4 Coding Questions (50 pts)

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 $\sum T_i$ 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`

1.5 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`.

1.6 Writeup Questions (15 pts)

(a) (3 pts) To compute the REINFORCE estimator, you will need to calculate the values $\{G_t\}_{t=1}^T$ (we drop the trajectory index i for simplicity), where

$$G_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$$

Naively, computing all these values takes $O(T^2)$ time. Describe how to compute them in $O(T)$ time.

(b) (12 pts) 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` should be `cartpole`, `pendulum`, or `cheetah`, and `SEED` should be a positive integer.

For each of the 3 environments, 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`). **Please include the plots (one for each environment) in your writeup, and comment on whether or not you observe improved performance when using a baseline.**

We have the following expectations about performance to receive full credit:

- 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)

2 Reducing Variance in Policy Gradient Methods (35 pts)

In class, we explored REINFORCE as a policy gradient method with no bias but high variance. In this problem, we will explore methods to dramatically reduce variance in policy gradient methods, potentially at the cost of increased bias.
Let us consider an infinite horizon MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma \rangle$. Let us define

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t) \tag{1}$$

An approximation to the policy gradient is defined as

$$g = \mathbb{E}_{\substack{s_{0:\infty} \\ a_{0:\infty}}}[\sum_{t=0}^{\infty} A^\pi(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t, s_t)] \tag{2}$$

where the colon notation $a : b$ represents the range $[a, a+1, a+2, ...b]$ inclusive of both ends.

(a) (5 pts) Let us define the partial sum $R_t = \sum_{i=0}^{t} r_i$. Show that it is not necessarily true that $\text{Var}(R_{t+1}) \geq \text{Var}(R_t)$. [Hint: Construct a counterexample MDP where this statement does not hold.]

(b) (10 pts) Prove that $\text{Var}(R_{t+1}) \geq \text{Var}(R_t)$ is true if we assume that r_{t+1} is, on average, correlated with the previous rewards, i.e. $\frac{1}{t+1}\sum_{i=0}^{t} \text{Cov}(r_i, r_{t+1}) > 0$.

(c) (5 pts) In practice, we do not have access to the true function $A^\pi(s_t, a_t)$, so we would like to obtain an estimate instead. We will consider the general form of an estimator $\hat{A}_t(s_{0:\infty}, a_{0:\infty})$ that can be a function of the entire trajectory.

Let $\hat{A}_t(s_{0:\infty}, a_{0:\infty}) = \hat{Q}_t(s_{t:\infty}, a_{t:\infty}) - b_t(s_{0:t}, a_{0:t-1})$, where for all s_t, a_t, we have that $\hat{Q}_t$ is an unbiased estimator of the true Q^π. Namely, we have that $\mathbb{E}_{\substack{s_{t+1:\infty} \\ a_{t+1:\infty}}}[\hat{Q}_t(s_{t:\infty}, a_{t:\infty})] = Q^\pi(s_t, a_t)$. Note that b_t is an arbitrary function of the actions and states sampled before a_t. Prove that by using this estimate of $\hat{A}_t$, we obtain an unbiased estimate of the policy gradient g. In other words, prove that $\mathbb{E}_{\substack{s_{0:\infty} \\ a_{0:\infty}}}[\sum_{t=0}^{\infty} \hat{A}_t(s_{0:\infty}, a_{0:\infty}) \nabla_\theta \log \pi_\theta(a_t, s_t)] = g$.

(d) (5 pts) We will now look at a few different variants of $\hat{A}_t$. Recall the TD error $\delta_t^{\hat{V}}(s_t, a_t) = r_t + \gamma \hat{V}(s_{t+1}) - \hat{V}(s_t)$. If $\hat{V} = V^\pi$, prove that $\delta_t^{\hat{V}}$ is an unbiased estimate of A^π.

(e) (5 pts) Let us define $\hat{A}_t^{(k)} = \sum_{i=0}^{k-1} \gamma^i \delta_{t+i}^{\hat{V}}$. Show that $\hat{A}_t^{(k)} = -\hat{V}(s_t) + \gamma^k \hat{V}(s_{t+k}) + \sum_{i=0}^{k-1} \gamma^i r_{t+i}$. In general, how does bias and variance change as k increases? (a few sentences of justification would suffice, no formal proof is necessary)

(f) (5 pts) Show that $\hat{A}_t^{(\infty)} = \sum_{i=0}^{\infty} \gamma^i r_{t+i} - \hat{V}(s_t)$.